

A Transformation-based Implementation for CLP with Qualification and Proximity *

R. CABALLERO, M. RODRÍGUEZ-ARTALEJO and C. A. ROMERO-DÍAZ

Departamento de Sistemas Informáticos y Computación, Universidad Complutense

Facultad de Informática, 28040 Madrid, Spain

(e-mail: {rafa,mario}@sip.ucm.es, cromdia@fdi.ucm.es)

submitted 26 July 2010; revised 19 March 2011, 7 January 2012; accepted 12 January 2012

Abstract

To appear in *Theory and Practice of Logic Programming (TPLP)*

Uncertainty in logic programming has been widely investigated in the last decades, leading to multiple extensions of the classical LP paradigm. However, few of these are designed as extensions of the well-established and powerful CLP scheme for Constraint Logic Programming. In a previous work we have proposed the SQCLP (*proximity-based qualified constraint logic programming*) scheme as a quite expressive extension of CLP with support for qualification values and proximity relations as generalizations of uncertainty values and similarity relations, respectively. In this paper we provide a transformation technique for transforming SQCLP programs and goals into semantically equivalent CLP programs and goals, and a practical Prolog-based implementation of some particularly useful instances of the SQCLP scheme. We also illustrate, by showing some simple—and working—examples, how the prototype can be effectively used as a tool for solving problems where qualification values and proximity relations play a key role. Intended use of SQCLP includes flexible information retrieval applications.

KEYWORDS: Constraint Logic Programming, Program Transformation, Qualification Domains and Values, Similarity and Proximity Relations, Flexible Information Retrieval.

1 Introduction

Many extensions of LP (*logic programming*) to deal with uncertain knowledge and uncertainty have been proposed in the last decades. These extensions have been proposed from different and somewhat unrelated perspectives, leading to multiple approaches in the way of using uncertain knowledge and understanding uncertainty.

A recent work by us (Rodríguez-Artalejo and Romero-Díaz 2010a) focuses on the declarative semantics of a new proposal for an extension of the CLP scheme supporting qualification values and proximity relations. More specifically, this work

* This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), PROMETIDOS-CM (S2009TIC-1465) and GPD-UCM (UCM-BSCH-GR58/08-910502).

defines a new generic scheme SQCLP (*proximity-based qualified constraint logic programming*) whose instances $SQCLP(\mathcal{S}, \mathcal{D}, \mathcal{C})$ are parameterized by a proximity relation \mathcal{S} , a qualification domain \mathcal{D} and a constraint domain \mathcal{C} . The current paper is intended as a continuation of (Rodríguez-Artalejo and Romero-Díaz 2010a) with the aim of providing a semantically correct program transformation technique that allows us to implement a sound and complete implementation of some useful instances of SQCLP on top of existing CLP systems like *SICStus Prolog* (SICS AB 2010) or *SWI-Prolog* (SWI-Prolog 2010). In the introductory section of (Rodríguez-Artalejo and Romero-Díaz 2010a) we have already summarized some related approaches of SQCLP with a special emphasis on their declarative semantics and their main semantic differences with SQCLP. In the next paragraphs we present a similar overview but, this time, putting the emphasis on the goal resolution procedures and system implementation techniques, when available.

Within the extensions of LP using annotations in program clauses we can find the seminal proposal of *quantitative logic programming* by (van Emden 1986) that inspired later works such as the GAP (*generalized annotated programs*) framework by (Kifer and Subrahmanian 1992) and our former scheme QLP (*qualified logic programming*). In the proposal of van Emden, one can find a primitive goal solving procedure based on and/or trees (these are similar to the alpha-beta trees used in game theory), used to prune the search space when proving some specific ground atom for some certainty value in the real interval $[0, 1]$. In the case of GAP, the goal solving procedure uses constrained SLD resolution in conjunction with a—costly—computation of so-called *reductants* between variants of program clauses. In contrast, QLP goal solving uses a more efficient resolution procedure called $SLD(\mathcal{D})$ resolution, implemented by means of real domain constraints, used to compute the qualification value of the head atom based on the attenuation factor of the program clause and the previously computed qualification values of the body atoms. Admittedly, the gain in efficiency of $SLD(\mathcal{D})$ w.r.t. GAP’s goal solving procedure is possible because QLP focuses on a more specialized class of annotated programs. While in all these three approaches there are some results of soundness and completeness, the results for the QLP scheme are the stronger ones (again, thanks to its also more focused scope w.r.t. GAP).

From a different viewpoint, extensions of LP supporting uncertainty can be roughly classified into two major lines: approaches based on fuzzy logic (Zadeh 1965; Hájek 1998; Gerla 2001) and approaches based on similarity relations. Historically, Fuzzy LP languages were motivated by expert knowledge representation applications. Early Fuzzy LP languages implementing the resolution principle introduced in (Lee 1972) include Prolog-Elf (Ishizuka and Kanai 1985), Fril Prolog (Baldwin et al. 1995) and F-Prolog (Li and Liu 1990). More recent approaches such as the Fuzzy LP languages in (Vojtáš 2001; Guadarrama et al. 2004) and Multi-Adjoint LP (MALP for short) in the sense of (Medina et al. 2001a) use clause annotations and a fuzzy interpretation of the connectives and aggregation operators occurring in program clauses and goals. The Fuzzy Prolog system proposed in (Guadarrama et al. 2004) is implemented by means of real constraints on top of a $CLP(\mathcal{R})$ system, using a syntactic expansion of the source code during the Prolog

compilation. A complete procedural semantics for MALP using reductants has been presented in (Medina et al. 2001b). A method for translating a MALP like program into standard Prolog has been described in (Julián et al. 2009).

The second line of research mentioned in the previous paragraph was motivated by applications in the field of flexible query answering. Classical LP is extended to Similarity-based LP (SLP for short), leading to languages which keep the classical syntax of LP clauses but use a similarity relation over a set of symbols S to allow “flexible” unification of syntactically different symbols with a certain approximation degree. Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as fuzzy relations represented by mappings $\mathcal{S} : S \times S \rightarrow [0, 1]$ which satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. Resolution with flexible unification can be used as a sound and complete goal solving procedure for SLP languages as shown e.g. in (Arcelli Fontana and Formato 2002; Sessa 2002). SLP languages include *Likelog* (Arcelli Fontana and Formato 1999; Arcelli Fontana 2002) and more recently *SiLog* (Loia et al. 2004), which has been implemented by means of an extended Prolog interpreter and proposed as a useful tool for web knowledge discovery.

In the last years, the SLP approach has been extended in various ways. The SQLP (*similarity-based qualified logic programming*) scheme proposed in (Caballero et al. 2008) extended SLP by allowing program clause annotations in QLP style and generalizing similarity relations to mappings $\mathcal{S} : S \times S \rightarrow D$ taking values in a qualification domain not necessarily identical to the real interval $[0, 1]$. As implementation technique for SQLP, (Caballero et al. 2008) proposed a semantically correct program transformation into QLP, whose goal solving procedure has been described above. Other related works on transformation-based implementations of SLP languages include (Sessa 2001; Medina et al. 2004). More recently, the SLP approach has been generalized to work with *proximity relations* in the sense of (Dubois and Prade 1980) represented by mappings $\mathcal{S} : S \times S \rightarrow [0, 1]$ which satisfy reflexivity and symmetry axioms but do not always satisfy transitivity. SLP like languages using proximity relations include *Bousi~Prolog* (Julián-Iranzo and Rubio-Manzano 2009a) and the SQCLP scheme (Rodríguez-Artalejo and Romero-Díaz 2010a). Two prototype implementations of *Bousi~Prolog* are available: a low-level implementation (Julián-Iranzo and Rubio-Manzano 2009b) based on an adaptation of the classical WAM (called *Similarity WAM*) implemented in JAVA and able to execute a Prolog program in the context of a similarity relation defined on the first order alphabet induced by that program; and a high-level implementation (Julián-Iranzo et al. 2009) done on top of *SWI-Prolog* by means of a program transformation from *Bousi~Prolog* programs into a so-called *Translated BPL code* than can be executed according to the weak SLD resolution principle by a meta-interpreter.

Let us now refer to approaches related to constraint solving and CLP. An analogy of proximity relations in the context of partial constraint satisfaction can be found in (Freuder and Wallace 1992), where several metrics are proposed to measure the proximity between the solution sets of two different constraint satisfaction problems. Moreover, some extensions of LP supporting uncertain reasoning use

constraint solving as implementation technique, as discussed in the previous paragraphs. However, we are only aware of three approaches which have been conceived as extensions of the classical CLP scheme proposed for the first time in (Jaffar and Lassez 1987). These three approaches are: (Riezler 1998) that extends the formulation of CLP by (Höhfeld and Smolka 1988) with quantitative LP in the sense of (van Emden 1986) and adapts van Emden’s idea of and/or trees to obtain a goal resolution procedure; (Bistarelli et al. 2001) that proposes a semiring-based approach to CLP, where constraints are solved in a soft way with levels of consistency represented by values of the semiring, and is implemented with `clp(FD,S)` for a particular class of semirings which enable to use local consistency algorithms, as described in (Georget and Codognet 1998); and the SQCLP scheme proposed in our previous work (Rodríguez-Artalejo and Romero-Díaz 2010a), which was designed as a common extension of SQLP and CLP.

As we have already said at the beginning of this introduction, this paper deals with transformation-based implementations of the SQCLP scheme. Our main results include: a) a transformation technique for transforming SQCLP programs into semantically equivalent CLP programs via two specific program transformations named `elimS` and `elimD`; and b) a practical Prolog-based implementation which relies on the aforementioned program transformations and supports several useful SQCLP instances. As far as we know, no previous work has dealt with the implementation of extended LP languages for uncertain reasoning which are able to support clause annotations, proximity relations and CLP style programming. In particular, our previous paper (Caballero et al. 2008) only presented a transformation analogous to `elimS` for a programming scheme less expressive than SQCLP, which supported neither non-transitive proximity relations nor CLP programming. Moreover, the transformation-based implementation reported in (Caballero et al. 2008) was not implemented in a system.

The reader is assumed to be familiar with the semantic foundations of LP (Lloyd 1987; Apt 1990) and CLP (Jaffar and Lassez 1987; Jaffar et al. 1998). The rest of the paper is structured as follows: Section 2 gives an abridged presentation of the SQCLP scheme and its declarative semantics, followed by an abstract discussion of goal solving intended to serve as a theoretical guideline for practical implementations. Section 3 briefly discusses two specializations of SQCLP, namely QCLP and CLP, which are used as the targets of the program transformations `elimS` and `elimD`, respectively. Section 4 presents these two program transformations along with mathematical results which prove their semantic correctness, relying on the declarative semantics of the SQCLP, QCLP and CLP schemes. Section 5 presents a Prolog-based prototype system that relies on the transformations proposed in the previous section and implements several useful SQCLP instances. Finally, Section 6 summarizes conclusions and points to some lines of planned future research.

2 The Scheme SQCLP and its Declarative Semantics

In this section we first recall the essentials of the SQCLP scheme and its declarative semantics, which were developed in detail in previous works (Rodríguez-Artalejo

and Romero-Díaz 2010a; Rodríguez-Artalejo and Romero-Díaz 2010b). Next we present an abstract discussion of goal solving intended to serve as a theoretical guideline for practical implementations of SQCLP instances.

2.1 Constraint Domains

As in the CLP scheme, we will work with constraint domains related to signatures. We assume an *universal programming signature* $\Gamma = \langle DC, DP \rangle$ where $DC = \bigcup_{n \in \mathbb{N}} DC^n$ and $DP = \bigcup_{n \in \mathbb{N}} DP^n$ are countably infinite and mutually disjoint sets of free function symbols (called *data constructors* in the sequel) and *defined predicate* symbols, respectively, ranked by arities. We will use *domain specific signatures* $\Sigma = \langle DC, DP, PP \rangle$ extending Γ with a disjoint set $PP = \bigcup_{n \in \mathbb{N}} PP^n$ of *primitive predicate* symbols, also ranked by arities. The idea is that primitive predicates come along with constraint domains, while defined predicates are specified in user programs. Each PP^n may be any countable set of n -ary predicate symbols.

Constraint domains \mathcal{C} , sets of constraints Π and their solutions, as well as terms, atoms and substitutions over a given \mathcal{C} are well known notions underlying the CLP scheme. In this paper we assume a relational formalization of constraint domains as mathematical structures \mathcal{C} providing a carrier set $C_{\mathcal{C}}$ (consisting of ground terms built from data constructors and a given set $B_{\mathcal{C}}$ of \mathcal{C} -specific basic values) and an interpretation of various \mathcal{C} -specific primitive predicates. For the examples in this paper we will use a constraint domain \mathcal{R} which allows to work with arithmetic constraints over the real numbers, and is defined to include:

- The set of basic values $B_{\mathcal{R}} = \mathbb{R}$. Note that $C_{\mathcal{R}}$ includes ground terms built from real values and data constructors, in addition to real numbers.
- Primitive predicates for encoding the usual arithmetic operations over \mathbb{R} . For instance, the addition operation $+$ over \mathbb{R} is encoded by a ternary primitive predicate op_+ such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $op_+(t_1, t_2, t)$ is true in \mathcal{R} iff $t_1, t_2, t \in \mathbb{R}$ and $t_1 + t_2 = t$. In particular, $op_+(t_1, t_2, t)$ is false in \mathcal{R} if either t_1 or t_2 includes data constructors. The primitive predicates encoding other arithmetic operations such as \times and $-$ are defined analogously.
- Primitive predicates for encoding the usual inequality relations over \mathbb{R} . For instance, the ordering \leq over \mathbb{R} is encoded by a binary primitive predicate cp_{\leq} such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $cp_{\leq}(t_1, t_2)$ is true in \mathcal{R} iff $t_1, t_2 \in \mathbb{R}$ and $t_1 \leq t_2$. In particular, $cp_{\leq}(t_1, t_2)$ is false in \mathcal{R} if either t_1 or t_2 includes data constructors. The primitive predicates encoding the other inequality relations, namely $>$, \geq and $<$, are defined analogously.

We assume the following classification of atomic \mathcal{C} -constraints: defined atomic constraints $p(\bar{t}_n)$, where p is a program-defined predicate symbol; primitive constraints $r(\bar{t}_n)$ where r is a \mathcal{C} -specific primitive predicate symbol; and equations $t == s$.

We use $\text{Conc}_{\mathcal{C}}$ as a notation for the set of all \mathcal{C} -constraints and κ as a notation for an atomic primitive constraint. Constraints are interpreted by means of \mathcal{C} -valuations $\eta \in \text{Val}_{\mathcal{C}}$, which are ground substitutions. The set $\text{Sol}_{\mathcal{C}}(\Pi)$ of solutions of $\Pi \subseteq \text{Conc}_{\mathcal{C}}$ includes all the valuations η such that $\Pi\eta$ is true when interpreted in \mathcal{C} . $\Pi \subseteq \text{Conc}_{\mathcal{C}}$

is called *satisfiable* if $\text{Sol}_C(\Pi) \neq \emptyset$ and *unsatisfiable* otherwise. $\pi \in \text{Con}_C$ is *entailed* by $\Pi \subseteq \text{Con}_C$ (noted $\Pi \models_C \pi$) iff $\text{Sol}_C(\Pi) \subseteq \text{Sol}_C(\pi)$.

2.2 Qualification Domains

Qualification domains were inspired by (van Emden 1986) and firstly introduced in (Rodríguez-Artalejo and Romero-Díaz 2008) with the aim of providing elements, called qualification values, which can be attached to computed answers. They are defined as structures $\mathcal{D} = \langle D, \leq, \mathbf{b}, \mathbf{t}, \circ \rangle$ verifying the following requirements:

1. $\langle D, \leq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points \mathbf{b} (called *infimum* or *bottom* element) and \mathbf{t} (called *maximum* or *top* element) w.r.t. the partial ordering \leq (called *qualification ordering*). For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e , and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e . We also write $d \triangleleft e$ as abbreviation for $d \leq e \wedge d \neq e$.
2. $\circ : D \times D \rightarrow D$, called *attenuation operation*, verifies the following axioms:
 - (a) \circ is associative, commutative and monotonic w.r.t. \leq .
 - (b) $\forall d \in D : d \circ \mathbf{t} = d$ and $d \circ \mathbf{b} = \mathbf{b}$.
 - (c) $\forall d, e \in D : d \circ e \leq e$.
 - (d) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = (d \circ e_1) \sqcap (d \circ e_2)$.

For any $S = \{e_1, e_2, \dots, e_n\} \subseteq D$, the *glb* (also called *infimum* of S) exists and can be computed as $\sqcap S = e_1 \sqcap e_2 \sqcap \dots \sqcap e_n$ (which reduces to \mathbf{t} in the case $n = 0$). The dual claim concerning *lubs* is also true. As an easy consequence of the axioms, one gets the identity $d \circ \sqcap S = \sqcap \{d \circ e \mid e \in S\}$.

Some of the axioms postulated for the attenuation operator—associativity, commutativity and monotonicity—are also required for t-norms in fuzzy logic, usually defined as binary operations over the real number interval $[0, 1]$. More generally, there are formal relationships between qualification domains and some other existing proposals of lattice-based structures for uncertain reasoning, such as the lower bound constraint frames proposed in (Gerla 2001), the multi-adjoint lattices for fuzzy LP languages proposed in (Medina et al. 2001a; Medina et al. 2001b) and the semirings for soft constraint solving proposed in (Bistarelli et al. 2001; Georget and Codognet 1998). However, qualification domains are a class of mathematical structures that differs from all these approaches. Their base lattices do not need to be complete and the axioms concerning the attenuation operator require additional properties w.r.t. t-norms. Some differences w.r.t. multi-adjoint algebras and the semirings from (Bistarelli et al. 2001) have been discussed in more detail in (Caballero et al. 2008) and (Rodríguez-Artalejo and Romero-Díaz 2010a), respectively.

Many useful qualification domains are such that $\forall d, e \in D \setminus \{\mathbf{b}\} : d \circ e \neq \mathbf{b}$. In the sequel, any qualification domain \mathcal{D} that verifies this property will be called *stable*. More technical details, explanations and examples concerning qualification domains can be found in (Rodríguez-Artalejo and Romero-Díaz 2010b). Examples include three basic qualification domains which are stable, namely: the qualification domain \mathcal{B} of classical boolean values, the qualification domain \mathcal{U} of uncertainty values, the qualification domain \mathcal{W} of weight values. Moreover, Theorem 2.1 of

(Rodríguez-Artalejo and Romero-Díaz 2010b) shows that the ordinary cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ of two qualification domains is again a qualification domain, while the strict cartesian product $\mathcal{D}_1 \otimes \mathcal{D}_2$ of two stable qualification domains is a stable qualification domain.

2.3 Expressing a Qualification Domain in a Constraint Domain

The SQCLP scheme depends crucially on the ability to encode qualification domains into constraint domains, in the sense defined below:

Definition 2.1 (Expressing \mathcal{D} in \mathcal{C})

A qualification domain \mathcal{D} is expressible in a constraint domain \mathcal{C} if there is an injective mapping $\iota : D \setminus \{\mathbf{b}\} \rightarrow C$ (thought as an embedding of $D \setminus \{\mathbf{b}\}$ into C) and moreover:

1. There is a \mathcal{C} -constraint $\mathbf{qVal}(X)$ with free variable X such that $\text{Sol}_{\mathcal{C}}(\mathbf{qVal}(X))$ is the set of all $\eta \in \text{Val}_{\mathcal{C}}$ verifying $\eta(X) \in \text{ran}(\iota)$.
Informal explanation: For each qualification value $x \in D \setminus \{\mathbf{b}\}$ we think of $\iota(x) \in C$ as the representation of x in \mathcal{C} . Therefore, $\text{ran}(\iota)$ is the set of those elements of C which can be used to represent qualification values, and $\mathbf{qVal}(X)$ constraints the value of X to be some of these representations.
2. There is a \mathcal{C} -constraint $\mathbf{qBound}(X, Y, Z)$ with free variables X, Y and Z encoding “ $x \trianglelefteq y \circ z$ ” in the following sense: any $\eta \in \text{Val}_{\mathcal{C}}$ such that $\eta(X) = \iota(x)$, $\eta(Y) = \iota(y)$ and $\eta(Z) = \iota(z)$ verifies $\eta \in \text{Sol}_{\mathcal{C}}(\mathbf{qBound}(X, Y, Z))$ iff $x \trianglelefteq y \circ z$.
Informal explanation: $\mathbf{qBound}(X, Y, Z)$ constraints the values of X, Y, Z to be the representations of three qualification values $x, y, z \in D \setminus \{\mathbf{b}\}$ such that $x \trianglelefteq y \circ z$.

In addition, if $\mathbf{qVal}(X)$ and $\mathbf{qBound}(X, Y, Z)$ can be chosen as existential constraints of the form $\exists X_1 \dots \exists X_n (B_1 \wedge \dots \wedge B_m)$ —where B_j ($1 \leq j \leq m$) are atomic—we say that \mathcal{D} is *existentially expressible* in \mathcal{C} . \square

It can be proved that $\mathcal{B}, \mathcal{U}, \mathcal{W}$ and any qualification domain built from these with the help of the strict cartesian product \otimes are existentially expressible in any constraint domain \mathcal{C} that includes the basic values and computational features of \mathcal{R} . The example below illustrates the existential representation of three typical qualification domains in \mathcal{R} :

Example 2.1

1. \mathcal{U} can be existentially expressed in \mathcal{R} as follows: $D_{\mathcal{U}} \setminus \{\mathbf{b}\} = D_{\mathcal{U}} \setminus \{0\} = (0, 1] \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$; therefore ι can be taken as the identity embedding mapping from $(0, 1]$ into \mathbb{R} . Moreover, $\mathbf{qVal}(X)$ can be built as the existential \mathcal{R} -constraint $cp_{<}(0, X) \wedge cp_{\leq}(X, 1)$ and $\mathbf{qBound}(X, Y, Z)$ can be built as the existential \mathcal{R} -constraint $\exists X'(op_{\times}(Y, Z, X') \wedge cp_{\leq}(X, X'))$.
2. \mathcal{W} can be existentially expressed in \mathcal{R} as follows: $D_{\mathcal{W}} \setminus \{\mathbf{b}\} = D_{\mathcal{W}} \setminus \{\infty\} = [0, \infty) \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$; therefore ι can be taken as the identity embedding mapping from $[0, \infty)$ into \mathbb{R} . Moreover, $\mathbf{qVal}(X)$ can be built as the existential \mathcal{R} -constraint $cp_{\geq}(X, 0)$ and $\mathbf{qBound}(X, Y, Z)$ can be built as the existential \mathcal{R} -constraint $\exists X'(op_{+}(Y, Z, X') \wedge cp_{\geq}(X, X'))$.

3. $\mathcal{U} \otimes \mathcal{W}$ can be existentially expressed in \mathcal{R} as follows: $D_{\mathcal{U} \otimes \mathcal{W}} \setminus \{\mathbf{b}\} = (0, 1] \times [0, \infty) \subseteq \mathbb{R} \times \mathbb{R}$; therefore $\iota : D_{\mathcal{U} \otimes \mathcal{W}} \setminus \{\mathbf{b}\} \rightarrow D_{\mathcal{R}}$ can be defined as $\iota(x, y) = \text{pair}(x, y)$, using a binary constructor $\text{pair} \in DC^2$ to represent the ordered pair (x, y) as an element of $D_{\mathcal{R}}$. Moreover, taking into account the two previous items of the example:

- $\mathbf{qVal}(X)$ can be built as $\exists X_1 \exists X_2 (X == \text{pair}(X_1, X_2) \wedge cp_{<}(0, X_1) \wedge cp_{\leq}(X_1, 1) \wedge cp_{\geq}(X_2, 0))$.
- $\mathbf{qBound}(X, Y, Z)$ can be built as $\exists X_1 \exists X'_1 \exists X_2 \exists X'_2 \exists Y_1 \exists Y_2 \exists Z_1 \exists Z_2 (X == \text{pair}(X_1, X_2) \wedge Y == \text{pair}(Y_1, Y_2) \wedge Z == \text{pair}(Z_1, Z_2) \wedge op_{\times}(Y_1, Z_1, X'_1) \wedge cp_{\leq}(X_1, X'_1) \wedge op_{+}(Y_2, Z_2, X'_2) \wedge cp_{\geq}(X_2, X'_2))$. \square

2.4 Programs and Declarative Semantics

Instances $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ of the SQCLP scheme are parameterized by so-called *admissible triples* $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ consisting of a constraint domain \mathcal{C} , a qualification domain \mathcal{D} and a proximity relation $\mathcal{S} : S \times S \rightarrow D$ —where D is the carrier set of \mathcal{D} and S is the set of all variables, basic values and signature symbols available in \mathcal{C} —satisfying the following properties:

- $\forall x \in S : \mathcal{S}(x, x) = \mathbf{t}$ (reflexivity).
- $\forall x, y \in S : \mathcal{S}(x, y) = \mathcal{S}(y, x)$ (symmetry).
- \mathcal{S} restricted to \mathcal{Var} behaves as the identity — i.e. $\mathcal{S}(X, X) = \mathbf{t}$ for all $X \in \mathcal{Var}$ and $\mathcal{S}(X, Y) = \mathbf{b}$ for all $X, Y \in \mathcal{Var}$ such that $X \neq Y$.
- For any $x, y \in S$, $\mathcal{S}(x, y) \neq \mathbf{b}$ can happen only if:
 - $x = y$ are identical.
 - x and y are both: basic values; data constructor symbols with the same arity; or defined predicate symbols with the same arity.

In particular, $\mathcal{S}(p, p') \neq \mathbf{b}$ cannot happen if p and p' are syntactically different primitive predicate symbols.

A proximity relation \mathcal{S} is called *similarity* iff it satisfies the additional property $\forall x, y, z \in S : \mathcal{S}(x, z) \supseteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$ (transitivity). A given proximity relation \mathcal{S} can be extended to work over terms, atoms and other syntactic objects in an obvious way. The definition for the case of terms is as follows:

1. For any term t , $\mathcal{S}(t, t) = \mathbf{t}$.
2. For $X \in \mathcal{Var}$ and for any term t different from X , $\mathcal{S}(X, t) = \mathcal{S}(t, X) = \mathbf{b}$.
3. For any two data constructor symbols c and c' with different arities, $\mathcal{S}(c(\bar{t}_n), c'(\bar{t}'_m)) = \mathbf{b}$.
4. For any two data constructor symbols c and c' with the same arity, $\mathcal{S}(c(\bar{t}_n), c'(\bar{t}'_n)) = \mathcal{S}(c, c') \sqcap \mathcal{S}(t_1, t'_1) \sqcap \dots \sqcap \mathcal{S}(t_n, t'_n)$.

For the case of finite substitutions σ and θ whose domain is a subset of a finite set of variables $\{X_1, \dots, X_m\}$, $\mathcal{S}(\sigma, \theta)$ can be naturally defined as $\mathcal{S}(X_1\sigma, X_1\theta) \sqcap \dots \sqcap \mathcal{S}(X_m\sigma, X_m\theta)$.

A $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program is a set \mathcal{P} of *qualified program rules* (also called *qualified clauses*) $C : A \stackrel{\alpha}{\leftarrow} B_1 \# w_1, \dots, B_m \# w_m$, where A is a defined atom, $\alpha \in D \setminus \{\mathbf{b}\}$ is called the *attenuation factor* of the clause and each $B_j \# w_j$ ($1 \leq j \leq m$) is an atom B_j annotated with a so-called *threshold value* $w_j \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$. The intended meaning of C is as follows: if for all $1 \leq j \leq m$ one has $B_j \# e_j$ (meaning that B_j holds with qualification value e_j) for some $e_j \triangleright^? w_j$, then $A \# d$ (meaning that A holds with qualification value d) can be inferred for any $d \in D \setminus \{\mathbf{b}\}$ such that $d \trianglelefteq \alpha \circ \bigcap_{j=1}^m e_j$. By convention, $e_j \triangleright^? w_j$ means $e_j \triangleright w_j$ if $w_j \neq ?$ and is identically true otherwise. In practice threshold values equal to ‘?’ and attenuation values equal to \mathbf{t} can be omitted.

Figure 1 shows a simple $\text{SQCLP}(\mathcal{S}_s, \mathcal{U}, \mathcal{R})$ -program \mathcal{P}_s which illustrates the expressivity of the SQCLP scheme to deal with problems involving flexible information retrieval. Predicate *search* can be used to answer queries asking for books in the library matching some desired language, genre and reader level. Predicate *guessRdrLvl* takes advantage of attenuation factors to encode heuristic rules to compute reader levels on the basis of vocabulary level and other book features. The other predicates compute book features in the natural way, and the proximity relation \mathcal{S}_s allows flexibility in any unification (i.e. solving of equality constraints) arising during the invocation of the program predicates.

The declarative semantics of a given $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} relies on *qualified constrained atoms* (briefly *qc-atoms*) of the form $A \# d \leftarrow \Pi$, intended to assert that the validity of atom A with qualification degree $d \in D$ is entailed by the constraint set Π . A qc-atom is called *defined*, *primitive* or *equational* according to the syntactic form of A ; and it is called *observable* iff $d \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable.

Program interpretations are defined as sets of observable qc-atoms which obey a natural closure condition. The results proved in (Rodríguez-Artalejo and Romero-Díaz 2010a) show two equivalent ways to characterize declarative semantics: using a fix-point approach and a proof-theoretical approach. For the purposes of the present paper it suffices to consider the proof-theoretical approach that relies on a formal inference system called *Proximity-based Qualified Constrained Horn Logic*—in symbols, $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ —intended to infer observable qc-atoms from \mathcal{P} and consisting of the three inference rules displayed in Figure 2. Rule **SQEA** depends on a relation $\approx_{d, \Pi}$ between terms that is defined in the following way: $t \approx_{d, \Pi} s$ iff there exist two terms \hat{t} and \hat{s} such that $\Pi \models_{\mathcal{C}} t == \hat{t}$, $\Pi \models_{\mathcal{C}} s == \hat{s}$ and $\mathbf{b} \neq d \trianglelefteq \mathcal{S}(\hat{t}, \hat{s})$. Recall that the notation $\Pi \models_{\mathcal{C}} \pi$ makes sense for any \mathcal{C} -constraint π and is a shorthand for $\text{Sol}_{\mathcal{C}}(\Pi) \subseteq \text{Sol}_{\mathcal{C}}(\pi)$, as explained in Subsection 2.1. The relation $\approx_{d, \Pi}$ allows to deduce equations from Π in a flexible way, i.e. taking the proximity relation \mathcal{S} into account. In the sequel we will use $t \approx_d s$ as a shorthand for $t \approx_{d, \emptyset} s$, which holds iff $\mathbf{b} \neq d \trianglelefteq \mathcal{S}(t, s)$.

We write $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \varphi$ to indicate that φ can be deduced from \mathcal{P} in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$, and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}}^k \varphi$ in the case that the deduction can be performed with exactly k **SQDA** inference steps. As usual in formal inference systems, $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proofs can be represented as *proof trees* whose nodes correspond to qc-atoms, each

```

% Book representation: book( ID, Title, Author, Lang, Genre, VocLvl, Pages ).
1 library([ book(1, 'Tintin', 'Hergé', french, comic, easy, 65),
2          book(2, 'Dune', 'F.P. Herbert', english, sciFi, medium, 345),
3          book(3, 'Kritik der reinen Vernunft', 'I. Kant', german, philosophy, difficult, 1011),
4          book(4, 'Beim Hauten der Zwiebel', 'G. Grass', german, biography, medium, 432) ])

% Auxiliary predicate for computing list membership:
5 member(B, [B|_])
6 member(B, [_|T]) ← member(B, T)

% Predicates for getting the explicit attributes of a given book:
7 getId(book(ID, _Title, _Author, _Lang, _Genre, _VocLvl, _Pages), ID)
8 getTitle(book(_ID, Title, _Author, _Lang, _Genre, _VocLvl, _Pages), Title)
9 getAuthor(book(_ID, _Title, Author, _Lang, _Genre, _VocLvl, _Pages), Author)
10 getLanguage(book(_ID, _Title, _Author, Lang, _Genre, _VocLvl, _Pages), Lang)
11 getGenre(book(_ID, _Title, _Author, _Lang, Genre, _VocLvl, _Pages), Genre)
12 getVocLvl(book(_ID, _Title, _Author, _Lang, _Genre, VocLvl, _Pages), VocLvl)
13 getPages(book(_ID, _Title, _Author, _Lang, _Genre, _VocLvl, Pages), Pages)

% Function for guessing the reader level of a given book:
14 guessRdrLvl(B, basic) ← getVocLvl(B, easy), getPages(B, N), N < 50
15 guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, easy), getPages(B, N), N ≥ 50
16 guessRdrLvl(B, basic) ←0.9 getGenre(B, children)
17 guessRdrLvl(B, proficiency) ←0.9 getVocLvl(B, difficult), getPages(B, N), N ≥ 200
18 guessRdrLvl(B, upper) ←0.8 getVocLvl(B, difficult), getPages(B, N), N < 200
19 guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, medium)
20 guessRdrLvl(B, upper) ←0.7 getVocLvl(B, medium)

% Function for answering a particular kind of user queries:
21 search(Lang, Genre, Level, Id) ← library(L)#1.0, member(B, L)#1.0,
22     getLanguage(B, Lang), getGenre(B, Genre),
23     guessRdrLvl(B, Level), getId(B, Id)#1.0

% Proximity relation  $\mathcal{S}_s$ :
24  $\mathcal{S}_s(\text{sciFi}, \text{fantasy}) = \mathcal{S}_s(\text{fantasy}, \text{sciFi}) = 0.9$ 
25  $\mathcal{S}_s(\text{adventure}, \text{fantasy}) = \mathcal{S}_s(\text{fantasy}, \text{adventure}) = 0.7$ 
26  $\mathcal{S}_s(\text{essay}, \text{philosophy}) = \mathcal{S}_s(\text{philosophy}, \text{essay}) = 0.8$ 
27  $\mathcal{S}_s(\text{essay}, \text{biography}) = \mathcal{S}_s(\text{biography}, \text{essay}) = 0.7$ 

```

Fig. 1. SQCLP($\mathcal{S}_s, \mathcal{U}, \mathcal{R}$)-program \mathcal{P}_s (*Library with books in different languages*)

node being inferred from its children by means of some SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) inference step.

The following theorem, proved in (Rodríguez-Artalejo and Romero-Díaz 2010b), characterizes least program models in the scheme SQCLP. This result allows to use SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-derivability as a logical criterion for proving the semantic correctness of program transformations, as we will do in Section 4.

$$\begin{array}{l}
\textbf{SQDA} \quad \frac{(\ (t'_i == t_i \theta) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \\
\\
\text{if } (p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}, \ \theta \text{ subst.}, \ \mathcal{S}(p', p) = d_0 \neq \mathbf{b}, \\
e_j \triangleright^? w_j \ (1 \leq j \leq m) \text{ and } d \trianglelefteq \prod_{i=0}^n d_i \sqcap \alpha \circ \prod_{j=1}^m e_j. \\
\\
\textbf{SQEA} \quad \frac{}{(t == s) \# d \Leftarrow \Pi} \quad \text{if } t \approx_{d, \Pi} s. \quad \textbf{SQPA} \quad \frac{}{\kappa \# d \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
\end{array}$$

Fig. 2. Proximity-based Qualified Constrained Horn Logic

Theorem 2.1 (Logical characterization of least program models in SQCHL)

For any SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \varphi\} \quad \square$$

2.5 Goals and Goal Solving

Goals for a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} have the form

$$G : A_1 \# W_1, \dots, A_m \# W_m \parallel W_1 \triangleright^? \beta_1, \dots, W_m \triangleright^? \beta_m$$

abbreviated as $(A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$. The $A_i \# W_i$ are called *annotated atoms*. If all atoms $A_i, i = 1 \dots m$, are equations $t_i == s_i$, the goal G is called a *unification problem*. The pairwise different variables $W_i \in \mathcal{W}\text{ar}$ are called qualification variables; they are taken from a set $\mathcal{W}\text{ar}$ assumed to be disjoint from the set $\mathcal{V}\text{ar}$ of data variables used in terms. The conditions $W_i \triangleright^? \beta_i$ (with $\beta_i \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$) are called *threshold conditions* and their intended meaning (relying on the notations ‘?’ and ‘ $\triangleright^?$ ’) is as already explained when introducing program clauses in Subsection 2.4. In the sequel, $\text{war}(o)$ will denote the set of all qualification variables occurring in the syntactic object o . In particular, for a goal G as displayed above, $\text{war}(G)$ denotes the set $\{W_i \mid 1 \leq i \leq m\}$. In the case $m = 1$ the goal is called *atomic*. The following definition relies on SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-derivability to provide a natural declarative notion of goal solution:

Definition 2.2 (Possible Answers and Goal Solutions)

Assume a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} and a goal G for \mathcal{P} with the syntax displayed above. Then:

1. A *possible answer* for G is any triple $\text{ans} = \langle \sigma, \mu, \Pi \rangle$ such that σ is a \mathcal{C} -substitution, $W\mu \in D \setminus \{\mathbf{b}\}$ for all $W \in \text{dom}(\mu)$, and Π is a satisfiable and finite set of atomic \mathcal{C} -constraints. The qualification value $\lambda_{\text{ans}} = \prod_{i=1}^m W_i \mu$ is called the *qualification level* of ans .
2. A possible answer $\langle \sigma, \mu, \Pi \rangle$ is called a *solution* for G iff the conditions $W_i \mu = d_i \triangleright^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$ hold for all $i = 1 \dots m$. Note that $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$ amounts to $t_i \sigma \approx_{W_i \mu, \Pi} s_i \sigma$ in the case that A_i is an equation $t_i == s_i$. The set of all solutions for G w.r.t. \mathcal{P} is noted $\text{Sol}_{\mathcal{P}}(G)$.

3. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \text{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i \eta$ is a ground atom for all $i = 1 \dots m$. The set of all ground solutions for G w.r.t. \mathcal{P} is noted $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
4. A ground solution $gsol = \langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by a possible answer $ans = \langle \sigma, \mu, \Pi \rangle$ iff $W_i \mu \geq W_i \rho$ for $i = 1 \dots m$ (which implies $\lambda_{ans} \geq \lambda_{gsol}$) and there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ such that $X\eta = X\sigma\nu$ holds for each variable $X \in \text{var}(G)$.
5. A ground solution $gsol = \langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by a possible answer $ans = \langle \sigma, \mu, \Pi \rangle$ *in the flexible sense* iff $\lambda_{ans} \geq \lambda_{gsol}$ and there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ such that $\mathcal{S}(X\eta, X\sigma\nu) \geq \lambda_{gsol}$ holds for each variable $X \in \text{var}(G)$. \square

A possible goal G_s for the library program displayed in Figure 1 is

$$G_s : \text{search}(\text{german}, \text{essay}, \text{intermediate}, ID) \# W \parallel W \geq 0.65$$

and one solution for G_s is $\langle \{ID \mapsto 4\}, \{W \mapsto 0.7\}, \emptyset \rangle$. In this simple case, the constraint set Π within the solution is empty.

The following example will be used to discuss some implementation issues in Subsection 5.1.3.

Example 2.2

Assume the admissible triple $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$ where the proximity relation \mathcal{S} is such that: $\mathcal{S}(a, b) = \mathcal{S}(b, a) = 0.9$, $\mathcal{S}(a, c) = \mathcal{S}(c, a) = 0.9$, and $\mathcal{S}(b, c) = \mathcal{S}(c, b) = 0.4$. Let \mathcal{P} be the empty program. Then, the goal G :

$$(X == Y) \# W_1, (X == b) \# W_2, (Y == c) \# W_3 \parallel W_1 \geq 0.8, W_2 \geq 0.8, W_3 \geq 0.8$$

is a unification problem. Its valid solutions in the sense of Definition 2.2 include $\text{sol}_i = \langle \sigma_i, \mu_i, \emptyset \rangle$ ($i = 1, 2, 3$), where:

$$\begin{array}{ll} \sigma_1 = \{X \mapsto a, Y \mapsto a\} & \mu_1 = \{W_1 \mapsto 1, W_2 \mapsto 0.9, W_3 \mapsto 0.9\} \\ \sigma_2 = \{X \mapsto b, Y \mapsto a\} & \mu_2 = \{W_1 \mapsto 0.9, W_2 \mapsto 1, W_3 \mapsto 0.9\} \\ \sigma_3 = \{X \mapsto a, Y \mapsto c\} & \mu_3 = \{W_1 \mapsto 0.9, W_2 \mapsto 0.9, W_3 \mapsto 1\} \end{array}$$

as well as some less interesting solutions assigning lower qualification values to the variables W_i ($i = 1, 2, 3$). In this simple example, all the solutions are ground, but this is not always the case in general. Note that sol_2 is subsumed by sol_1 in the flexible sense because:

- $\nu = \varepsilon \in \text{Sol}_{\mathcal{C}}(\emptyset)$ satisfies $\mathcal{S}(X\sigma_2, X\sigma_1\varepsilon) = \mathcal{S}(b, a) = 0.9 \geq 0.9$ and also $\mathcal{S}(Y\sigma_2, Y\sigma_1\varepsilon) = \mathcal{S}(a, a) = 1 \geq 0.9$.
- The qualification level of both sol_2 and sol_1 is 0.9, thus trivially, $0.9 \geq 0.9$.

Moreover, sol_3 is also subsumed by sol_1 in the flexible sense, because:

- $\nu = \varepsilon \in \text{Sol}_{\mathcal{C}}(\emptyset)$ satisfies $\mathcal{S}(X\sigma_3, X\sigma_1\varepsilon) = \mathcal{S}(a, a) = 1 \geq 0.9$ and also $\mathcal{S}(Y\sigma_3, Y\sigma_1\varepsilon) = \mathcal{S}(c, a) = 0.9 \geq 0.9$.
- The qualification level of both sol_3 and sol_1 is 0.9, thus trivially, $0.9 \geq 0.9$.

In fact, it is easy to check that any of the three ground solutions sol_1 , sol_2 and sol_3 subsumes the other two in the flexible sense. \square

In practice, users of SQCLP languages will rely on some available *goal solving system* for computing goal solutions. The following definition provides an abstract specification of semantically correct goal solving systems which will serve as a theoretical guideline for the implementation presented in Section 5:

Definition 2.3 (Correct Abstract Goal Solving Systems for SQCLP)

An *abstract goal solving system* for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ is any device \mathcal{CA} that takes a program \mathcal{P} and a goal G as input and yields a set $\mathcal{CA}_{\mathcal{P}}(G)$ of possible answers $\langle \sigma, \mu, \Pi \rangle$ (called *computed answers*) as output. Moreover:

1. \mathcal{CA} is called *sound* iff every computed answer is a solution, i.e. $\mathcal{CA}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
2. \mathcal{CA} is called *weakly complete* iff for every ground solution $gsol \in \text{GSol}_{\mathcal{P}}(G)$ there is some computed answer $ans \in \mathcal{CA}_{\mathcal{P}}(G)$ such that ans subsumes $gsol$.
3. \mathcal{CA} is called *weakly complete in the flexible sense* iff for every ground solution $gsol \in \text{GSol}_{\mathcal{P}}(G)$ there is some computed answer $ans \in \mathcal{CA}_{\mathcal{P}}(G)$ such that ans subsumes $gsol$ in the flexible sense.
4. \mathcal{CA} is called *correct* iff it is both sound and weakly complete.
5. \mathcal{CA} is called *correct in the flexible sense* iff it is both sound and weakly complete in the flexible sense. \square

Extensions of the well-known SLD-resolution procedure (Lloyd 1987; Apt 1990) can be used as a basis to obtain correct goal solving systems for extended LP languages. In particular, constraint SLD-resolution provides a correct goal solving system for instances of the CLP scheme, as proved e.g. in (Jaffar et al. 1998)¹. Several extensions of the SLD-resolution, tailored to different LP languages supporting uncertain reasoning, have already been mentioned in Section 1.

Rather than developing an extension of SLD resolution tailored to the SQCLP scheme, our aim in this paper is to investigate goal solving systems based on a semantically correct program transformation from SQCLP into CLP. Sections 4 and 5 present the transformation technique and its implementation on top of a CLP **Prolog** system, respectively. As we will explain in Subsection 5.1, weak completeness as specified in Definition 2.3(2) is very hard to achieve in a practical implementation, while flexible weak completeness in the sense of Definition 2.3(3) is a satisfactory notion for extended LP languages which use proximity relations. For instance, similarity-based SLD resolution as presented in (Sessa 2002) is complete in a flexible sense. Therefore, the **Prolog**-based prototype system presented in Section 5 aims at soundness and weak completeness in the flexible sense, as specified in Definition 2.3(3). The definition and lemma below can be used as an abstract guideline for converting a correct goal solving system \mathcal{CA} into another goal solving system \mathcal{FCA} which is correct in the flexible sense and may be easier to implement, because it yields smaller sets of computed answers.

¹ In fact, constraint SLD-resolution is complete in a stronger sense than weak completeness. As proved in (Jaffar et al. 1998), every solution - even if it is not ground - is subsumed in a suitable sense by a finite set of computed solutions.

Definition 2.4 (Flexible Restrictions of an Abstract Goal Solving System)

Let \mathcal{CA} and \mathcal{FCA} be two abstract goal solving systems for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$. We say that \mathcal{FCA} is a *flexible restriction* of \mathcal{CA} iff the two following conditions hold for any choice of a program \mathcal{P} and a goal $G : (A_i \sharp W_i, W_i \triangleright^? \beta_i)_{i=1\dots m}$:

1. $\mathcal{FCA}_{\mathcal{P}}(G) \subseteq \mathcal{CA}_{\mathcal{P}}(G)$. Informally, \mathcal{FCA} is restricted to compute some of the answers computed by \mathcal{CA} .
2. For each $ans = \langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$ there is some $\widehat{ans} = \langle \hat{\sigma}, \hat{\mu}, \Pi \rangle \in \mathcal{FCA}_{\mathcal{P}}(G)$ such that $\lambda_{\widehat{ans}} \triangleright \lambda_{ans}$ and $\mathcal{S}(X\sigma, X\hat{\sigma}) \triangleright \lambda_{ans}$ holds for each variable $X \in \text{var}(G)$. Informally, each answer computed by \mathcal{CA} is close (w.r.t. \mathcal{S}) to some of the answers computed by \mathcal{FCA} . \square

Lemma 2.1 (Flexible Correctness of Flexible Restrictions)

Let \mathcal{CA} be a correct abstract goal solving system for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$. Then any flexible restriction \mathcal{FCA} of \mathcal{CA} is correct in the flexible sense.

Proof

By assumption, \mathcal{CA} is sound and weakly complete. We must prove soundness and weak completeness in the flexible sense for \mathcal{FCA} . Let a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} and a goal $G : (A_i \sharp W_i, W_i \triangleright^? \beta_i)_{i=1\dots m}$ for \mathcal{P} be given.

— *Soundness.* $\mathcal{FCA}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$ trivially follows from $\mathcal{FCA}_{\mathcal{P}}(G) \subseteq \mathcal{CA}_{\mathcal{P}}(G)$ (true because \mathcal{FCA} refines \mathcal{CA}) and $\mathcal{CA}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$ (true because \mathcal{CA} is sound).

— *Weak completeness in the flexible sense.* In order to check the conditions stated in Definition 2.3(3), let $gsol = \langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ be given. Since \mathcal{CA} is weakly complete, there is some $ans = \langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$ that subsumes $gsol$ and hence:

- (a) $W_i \mu \triangleright W_i \rho$ for $i = 1 \dots m$, which implies $\lambda_{ans} \triangleright \lambda_{gsol}$.
- (b) There is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ such that $X\eta = X\sigma\nu$ holds for all $X \in \text{var}(G)$.

Since \mathcal{FCA} is a flexible refinement of \mathcal{CA} , there is some $\widehat{ans} = \langle \hat{\sigma}, \hat{\mu}, \Pi \rangle \in \mathcal{FCA}_{\mathcal{P}}(G)$ that is close to ans and thus verifies:

- (c) $\lambda_{\widehat{ans}} \triangleright \lambda_{ans}$.
- (d) $\mathcal{S}(X\sigma, X\hat{\sigma}) \triangleright \lambda_{ans}$ holds for all $X \in \text{var}(G)$.

Now we can claim:

- (e) $\lambda_{\widehat{ans}} \triangleright \lambda_{gsol}$ — follows from (c) and (a).
- (f) $\mathcal{S}(X\sigma\nu, X\hat{\sigma}\nu) \triangleright \lambda_{gsol}$ holds for all $X \in \text{var}(G)$ — follows from (d) and (a).
- (g) $\mathcal{S}(X\eta, X\hat{\sigma}\nu) \triangleright \lambda_{gsol}$ holds for all $X \in \text{var}(G)$ — follows from (f) and (b).

Since $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$, (e) and (g) guarantee that \widehat{ans} subsumes $gsol$ in the flexible sense. This finishes the proof. \square

Let us finish this section with a remark concerning unification. Both our implementation and SLD-based goal solving systems for SLP languages—we view (Arcelli Fontana and Formato 2002; Sessa 2002) as representative proposals of this kind; others have been cited in Section 1—must share the ability to solve unification problems modulo a given proximity relation $\mathcal{S} : S \times S \rightarrow [0, 1]$ over signature

symbols, that is assumed to be transitive in (Sessa 2002) and some other related works, but not in Bousi~Prolog (Julián-Iranzo and Rubio-Manzano 2009a; Julián-Iranzo and Rubio-Manzano 2009b) and our own approach. The lack of transitivity makes a crucial difference. The unification algorithms modulo \mathcal{S} known for the case that \mathcal{S} is a similarity relation fail to be complete in the flexible sense if \mathcal{S} is a non-transitive proximity relation. More details on this issue are given in Subsection 5.1 when discussing the implementation of unification modulo \mathcal{S} in our prototype system for SQCLP programming.

3 The Schemes QCLP & CLP as Specializations of SQCLP

As discussed in the concluding section of (Rodríguez-Artalejo and Romero-Díaz 2010a), several specializations of the SQCLP scheme can be obtained by partial instantiation of its parameters. In particular, QCLP and CLP can be defined as schemes with instances:

$$\begin{aligned} \text{QCLP}(\mathcal{D}, \mathcal{C}) &=_{\text{def}} \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C}) \\ \text{CLP}(\mathcal{C}) &=_{\text{def}} \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{C}) = \text{QCLP}(\mathcal{B}, \mathcal{C}) \end{aligned}$$

with \mathcal{S}_{id} the *identity* proximity relation and \mathcal{B} the qualification domain including just the two classical boolean values. As explained in the introduction, QCLP and CLP are the targets of two program transformations to be developed in Section 4. In this brief section we provide an explicit description of the syntax and semantics of these two schemes, derived from their behaviour as specializations of SQCLP.

3.1 Presentation of the QCLP Scheme

As already explained, the instances of QCLP can be defined by the equation $\text{QCLP}(\mathcal{D}, \mathcal{C}) = \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C})$. Due to the admissibility of the parameter triple $\langle \mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C} \rangle$, the qualification domain \mathcal{D} must be (existentially) expressible in the constraint domain \mathcal{C} . Technically, the QCLP scheme can be seen as a common extension of the classical CLP scheme for Constraint Logic Programming (Jaffar and Lassez 1987; Jaffar et al. 1998) and the QLP scheme for Qualified Logic Programming originally introduced in (Rodríguez-Artalejo and Romero-Díaz 2008). Intuitively, QCLP programming behaves like SQCLP programming, except that proximity information other than the identity is not available for proving equalities.

Program clauses and observable qc-atoms in QCLP are defined in the same way as in SQCLP. The library program \mathcal{P}_s in Figure 1 becomes a $\text{QCLP}(\mathcal{U}, \mathcal{R})$ -program \mathcal{P}'_s just by replacing \mathcal{S}_{id} for \mathcal{S} . Of course, \mathcal{P}'_s does not support flexible unification as it was the case with \mathcal{P}_s .

As explained in Subsection 2.4, the proof system consisting of the three displayed in Figure 2 characterizes the declarative semantics of a given $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} . In the particular case $\mathcal{S} = \mathcal{S}_{\text{id}}$, the inference rules specialize to those displayed in Figure 3, yielding a formal proof system called *Qualified Constrained Horn Logic*—in symbols, $\text{QCHL}(\mathcal{D}, \mathcal{C})$ —which characterizes the declarative semantics of a given $\text{QCLP}(\mathcal{D}, \mathcal{C})$ -program \mathcal{P} . Note that rule **SQEA** depends on a relation

\approx_Π between terms that is defined to behave the same as the specialization of $\approx_{d,\Pi}$ to the case $\mathcal{S} = \mathcal{S}_{\text{id}}$. It is easily checked that $t \approx_\Pi s$ does not depend on d and holds iff $\Pi \models_{\mathcal{C}} t = s$. Both $\approx_{d,\Pi}$ and \approx_Π allow to use the constraints within Π when deducing equations. However, $c(\bar{t}_n) \approx_\Pi c'(\bar{s}_n)$ never holds in the case that c and c' are not syntactically identical.

$$\begin{array}{l}
\mathbf{QDA} \quad \frac{((t'_i = t_i \theta) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p(\bar{t}'_n) \# d \Leftarrow \Pi} \\
\text{if } (p(\bar{t}_n) \Leftarrow^\alpha B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}, \theta \text{ subst.}, \\
e_j \triangleright^? w_j \ (1 \leq j \leq m) \text{ and } d \trianglelefteq \prod_{i=1}^n d_i \sqcap \alpha \circ \prod_{j=1}^m e_j. \\
\\
\mathbf{QEA} \quad \frac{}{(t = s) \# d \Leftarrow \Pi} \quad \text{if } t \approx_\Pi s. \qquad \mathbf{QPA} \quad \frac{}{\kappa \# d \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
\end{array}$$

Fig. 3. Qualified Constrained Horn Logic

SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) proof trees and the notations related to them can be naturally specialized to QCHL(\mathcal{D}, \mathcal{C}). In particular, we will use the notation $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \varphi$ (resp. $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}}^k \varphi$) to indicate that the qc-atom φ can be inferred in QCHL(\mathcal{D}, \mathcal{C}) from the program \mathcal{P} (resp. it can be inferred by using exactly k **QDA** inference steps). Clearly, Theorem 2.1 specializes to QCHL yielding the following result that is stated here for convenience:

Theorem 3.1 (Logical characterization of least program models in QCHL)

For any QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \varphi\} \quad \square$$

Concerning goals and their solutions, their specialization to the particular case $\mathcal{S} = \mathcal{S}_{\text{id}}$ leaves the syntax of goals G unaffected and leads to the following definition, almost identical to Definition 2.2:

Definition 3.1 (Possible Answers and Goal Solutions in QCLP)

Assume a given QCLP(\mathcal{S}, \mathcal{D}) \mathcal{C} -program \mathcal{P} and a goal $G : (A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$. Then:

1. *Possible answers* $ans = \langle \sigma, \mu, \Pi \rangle$ for G and their qualification levels are defined as in SQCLP (see Definition 2.2(1)).
2. A *solution* for G is any possible answer $\langle \sigma, \mu, \Pi \rangle$ that verifies the conditions in Definition 2.2(2), except that the requirement $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$ used in Definition 2.2 for SQCLP becomes now $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$ for QCLP. The set of all solutions for G is noted $\text{Sol}_{\mathcal{P}}(G)$.
3. The subset $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$ of all *ground* solutions is defined exactly as in Definition 2.2(3).

4. The subsumption relation between a ground solution $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ and an arbitrary solution $\langle \sigma, \mu, \Pi \rangle$ is defined exactly as in Definition 2.2(4). Subsumption in the flexible sense cannot be considered in QCLP due to the absence of a proximity relation. \square

Finally, the notion of correct abstract goal solving system for SQCLP given in Definition 2.3 specializes to QCLP with only one minor modification: weak completeness in the flexible sense cannot be considered here, due to the absence of a proximity relation. Therefore, we state the following definition:

Definition 3.2 (Correct Abstract Goal Solving Systems for QCLP)

An *abstract goal solving system* for $\text{QCLP}(\mathcal{D}, \mathcal{C})$ is any device \mathcal{CA} that takes a program \mathcal{P} and a goal G as input and yields a set $\mathcal{CA}_{\mathcal{P}}(G)$ of possible answers $\langle \sigma, \mu, \Pi \rangle$ (called *computed answers*) as output. Moreover:

1. \mathcal{CA} is called *sound* iff every computed answer is a solution, i.e. $\mathcal{CA}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
2. \mathcal{CA} is called *weakly complete* iff for every ground solution $gsol \in \text{GSol}_{\mathcal{P}}(G)$ there is some computed answer $ans \in \mathcal{CA}_{\mathcal{P}}(G)$ such that ans subsumes $gsol$.
3. \mathcal{CA} is called *correct* iff it is both sound and weakly complete. \square

3.2 Presentation of the CLP Scheme

As already explained, the instances of CLP can be defined by the equation $\text{CLP}(\mathcal{C}) = \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{C})$, or equivalently, $\text{CLP}(\mathcal{C}) = \text{QCLP}(\mathcal{B}, \mathcal{C})$. Due to the fixed choice $\mathcal{D} = \mathcal{B}$, the only qualification value $d \in D \setminus \{\mathbf{b}\}$ available for use as attenuation factor or threshold value is $d = \mathbf{t}$. Therefore, CLP can only include threshold values equal to ‘?’ and attenuation values equal to the top element $\mathbf{t} = \text{true}$ of \mathcal{B} . As explained in Section 2, such trivial threshold and attenuation values can be omitted, and CLP clauses can be written with the simplified syntax $A \leftarrow B_1, \dots, B_m$.

Since $\mathbf{t} = \text{true}$ is the only non-trivial qualification value available in CLP, qc-atoms $A \sharp d \Leftarrow \Pi$ are always of the form $A \sharp \text{true} \Leftarrow \Pi$ and can be written as $A \Leftarrow \Pi$. Moreover, all the side conditions for the inference rule **QDA** in Figure 3 become trivial when specialized to the case $\mathcal{D} = \mathcal{B}$. Therefore, the specialization of $\text{QCHL}(\mathcal{D}, \mathcal{C})$ to the case $\mathcal{D} = \mathcal{B}$ leads to the formal proof system called *Constrained Horn Logic*—in symbols, $\text{CHL}(\mathcal{C})$ —consisting of the three inference rules displayed in Figure 4, which characterizes the declarative semantics of a given $\text{CLP}(\mathcal{C})$ -program \mathcal{P} .

$\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof trees and the notations related to them can be naturally specialized to $\text{CHL}(\mathcal{C})$. In particular, we will use the notation $\mathcal{P} \vdash_{\mathcal{C}} \varphi$ (resp. $\mathcal{P} \vdash_{\mathcal{C}}^k \varphi$) to indicate that the qc-atom φ can be inferred in $\text{CHL}(\mathcal{C})$ from the program \mathcal{P} (resp. it can be inferred by using exactly k **DA** inference steps). Clearly, Theorem 3.1 specializes to CHL yielding the following result that is stated here for convenience:

Theorem 3.2 (Logical characterization of least program models in CHL)

For any $\text{CLP}(\mathcal{C})$ -program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{C}} \varphi\} \quad \square$$

$$\begin{array}{l}
\mathbf{DA} \quad \frac{((t'_i == t_i \theta) \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \Leftarrow \Pi)_{j=1 \dots m}}{p(\bar{t}'_n) \Leftarrow \Pi} \\
\text{if } (p(\bar{t}'_n) \Leftarrow B_1, \dots, B_m) \in \mathcal{P} \text{ and } \theta \text{ subst.} \\
\mathbf{EA} \quad \frac{}{(t == s) \Leftarrow \Pi} \quad \text{if } t \approx_{\Pi} s. \qquad \mathbf{PA} \quad \frac{}{\kappa \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
\end{array}$$

Fig. 4. Constrained Horn Logic

Concerning goals and their solutions, their specialization to the scheme CLP leads to the following definition:

Definition 3.3 (Goals and their Solutions in CLP)

Assume a given $\text{CLP}(\mathcal{C})$ -program \mathcal{P} . Then:

1. Goals for \mathcal{P} have the form $G : A_1, \dots, A_m$, abbreviated as $(A_i)_{i=1 \dots m}$, where A_i ($1 \leq i \leq m$) are atoms.
2. A *possible answer* for a goal G is any pair $ans = \langle \sigma, \Pi \rangle$ such that σ is a \mathcal{C} -substitution and Π is a satisfiable and finite set of atomic \mathcal{C} -constraints.
3. A possible answer $\langle \sigma, \Pi \rangle$ is called a *solution* for G iff $\mathcal{P} \vdash_{\mathcal{C}} A_i \sigma \Leftarrow \Pi$ holds for all $i = 1 \dots m$. The set of all solutions for G is noted $\text{Sol}_{\mathcal{P}}(G)$.
4. A solution $\langle \eta, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \text{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i \eta$ is a ground atom for all $i = 1 \dots m$. The set of all ground solutions for G is noted $\text{GSol}_{\mathcal{P}}(G)$. Obviously, $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
5. A ground solution $\langle \eta, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \Pi \rangle$ iff there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\text{var}(G)} \sigma \nu$. \square

The notion of correct abstract goal solving system for SQCLP given in Definition 3.2 specializes to CLP with a minor change, namely: computed answers are pairs $\langle \sigma, \Pi \rangle$. Formally, the definition for CLP is as follows:

Definition 3.4 (Correct Abstract Goal Solving Systems for CLP)

A *goal solving system* for $\text{CLP}(\mathcal{C})$ is any device \mathcal{CA} that takes a program \mathcal{P} and a goal G as input and yields a set $\mathcal{CA}_{\mathcal{P}}(G)$ of possible answers $\langle \sigma, \Pi \rangle$ (called *computed answers*) as output. Moreover, soundness, weak completeness and weak correctness of \mathcal{CA} are defined exactly as in Definition 3.2. \square

We close this Subsection with a technical lemma that will be useful for proving some results in Subsection 4.2:

Lemma 3.1

Assume an existential \mathcal{C} -constraint $\pi(\bar{X}_n) = \exists Y_1 \dots \exists Y_k (B_1 \wedge \dots \wedge B_m)$ with free variables \bar{X}_n and a given $\text{CLP}(\mathcal{C})$ -program \mathcal{P} including the clause $C : p(\bar{X}_n) \Leftarrow B_1, \dots, B_m$, where $p \in DP^n$ does not occur at the head of any other clause of \mathcal{P} . Then, for any n-tuple \bar{t}_n of \mathcal{C} -terms and any finite and satisfiable $\Pi \subseteq \text{Con}_{\mathcal{C}}$, one has:

1. $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi) \implies \Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$, where $\pi(\bar{t}_n)$ stands for the result of applying the substitution $\{\bar{X}_n \mapsto \bar{t}_n\}$ to $\pi(\bar{X}_n)$.
2. The opposite implication $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \implies \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ holds if \bar{t}_n is a ground term tuple. Note that for ground \bar{t}_n the constraint entailment $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ simply means that $\pi(\bar{t}_n)$ is true in \mathcal{C} .

Proof

We prove each item separately:

1. Assume $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$. Note that C is the only clause for p in \mathcal{P} and that each atom B_j in C 's body is an atomic constraint. Therefore, the $\text{CHL}(\mathcal{C})$ proof must use a **DA** step based on an instance $C\theta$ of clause C such that $\Pi \models_{\mathcal{C}} t_i == X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models B_j\theta$ holds for all $1 \leq j \leq m$. These conditions and the syntactic form of $\pi(\bar{X}_n)$ obviously imply $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$.
2. Assume now $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ and \bar{t}_n ground. Then $\pi(\bar{t}_n)$ is true in \mathcal{C} , and due to the syntactic form of $\pi(\bar{X}_n)$, there must be some substitution θ such that $X_i\theta = t_i$ (syntactic identity) for all $1 \leq i \leq n$ and $B_j\theta$ is ground and true in \mathcal{C} for all $1 \leq j \leq m$. Trivially, $\Pi \models_{\mathcal{C}} t_i == X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models_{\mathcal{C}} B_j\theta$ also holds for all $1 \leq j \leq m$. Then, it is obvious that $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ can be proved by using a **DA** step based on the instance $C\theta$ of clause C . \square

We remark that the second item of the previous lemma can fail if \bar{t}_n is not ground. This can be checked by presenting a counterexample based on the constraint domain \mathcal{R} , using the syntax for \mathcal{R} -constraints explained in (Rodríguez-Artalejo and Romero-Díaz 2010b). Consider the existential \mathcal{R} -constraint $\pi(X) = \exists Y (op_+(Y, Y, X))$, and a $\text{CLP}(\mathcal{R})$ -program \mathcal{P} including the clause $C : p(X) \Leftarrow op_+(Y, Y, X)$ and no other occurrence of the defined predicate symbol p . Consider also $\Pi = \{cp_{\geq}(X, 0.0)\}$ and $t = X$. Then $\Pi \models_{\mathcal{R}} \pi(X)$ is obviously true, because any real number $x \geq 0.0$ satisfies $\exists Y (op_+(Y, Y, x))$ in \mathcal{R} . However, there is no \mathcal{R} -term s such that $\Pi \models_{\mathcal{R}} op_+(s, s, X)$, and therefore there is no instance $C\theta$ of clause C that can be used to prove $\mathcal{P} \vdash_{\mathcal{C}} (p(X) \Leftarrow \Pi)$ by applying a **DA** step.

4 Implementation by Program Transformation

The purpose of this section is to introduce a program transformation that transforms $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ programs and goals into semantically equivalent $\text{CLP}(\mathcal{C})$ programs and goals. This transformation is performed as the composition of the two following specific transformations:

1. $\text{elim}_{\mathcal{S}}$ — Eliminates the proximity relation \mathcal{S} of arbitrary $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ programs and goals, producing equivalent $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals.
2. $\text{elim}_{\mathcal{D}}$ — Eliminates the qualification domain \mathcal{D} of arbitrary $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals, producing equivalent $\text{CLP}(\mathcal{C})$ programs and goals.

Thus, given a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} —resp. $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -goal G —, the composition of the two transformations will produce an equivalent $\text{CLP}(\mathcal{C})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ —resp. $\text{CLP}(\mathcal{C})$ -goal $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G))$ —.

Example 4.1 (Running example: SQCLP($\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program \mathcal{P}_r)

As a running example for this section, consider the SQCLP($\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program \mathcal{P}_r as follows:

$$\begin{array}{ll}
R_1 & famous(sha) \xleftarrow{(0.9,1)} \\
R_2 & wrote(sha, kle) \xleftarrow{(1,1)} \\
R_3 & wrote(sha, hamlet) \xleftarrow{(1,1)} \\
R_4 & good_work(G) \xleftarrow{(0.75,3)} famous(A) \# (0.5, 100), authored(A, G) \\
s_1 & \mathcal{S}_r(wrote, authored) = \mathcal{S}_r(authored, wrote) = (0.9, 0) \\
s_2 & \mathcal{S}_r(kle, kli) = \mathcal{S}_r(kli, kle) = (0.8, 2)
\end{array}$$

where the constants *shakespeare*, *king_lear* and *king_liar* have been respectively replaced, for clarity purposes in the subsequent examples, by *sha*, *kle* and *kli*.

In addition, consider the SQCLP($\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-goal G_r as follows:

$$good_work(X) \# W \parallel W \triangleright^? (0.5, 10)$$

We will illustrate the two transformation by showing, in subsequent examples, the program clauses of $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ and the goals $\text{elim}_{\mathcal{S}}(G_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G_r))$. \square

In the following subsections we explain both transformations in detail and we show that they can be used to specify abstract goal solving systems for SQCLP.

4.1 Transforming SQCLP into QCLP

In this subsection we assume that the triple $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is admissible. In the sequel we say that a defined predicate symbol $p \in DP^n$ is *affected* by a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} iff $\mathcal{S}(p, p') \neq \mathbf{b}$ for some p' occurring in \mathcal{P} . We also say that an atom A is *relevant* for \mathcal{P} iff some of the three following cases hold: a) A is an equation $t == s$; b) A is a primitive atom κ ; or c) A is a defined atom $p(\bar{t}_n)$ such that p is affected by \mathcal{P} .

As a first step towards the definition of the first program transformation $\text{elim}_{\mathcal{S}}$, we define a set $EQ_{\mathcal{S}}$ of QCLP(\mathcal{D}, \mathcal{C}) program clauses that emulates the behaviour of equations in SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$). The following definition assumes that the binary predicate symbol $\sim \in DP^2$ (used in infix notation) and the nullary predicate symbols $\text{pay}_{\lambda} \in DP^0$ are not affected by \mathcal{P} .

Definition 4.1

We define $EQ_{\mathcal{S}}$ as the following QCLP(\mathcal{D}, \mathcal{C})-program:

$$\begin{aligned}
EQ_{\mathcal{S}} =_{\text{def}} \{ & X \sim Y \xleftarrow{\mathbf{t}} (X == Y) \# ? \} \\
& \cup \{ u \sim u' \xleftarrow{\mathbf{t}} \text{pay}_{\lambda} \# ? \mid u, u' \in B_{\mathcal{C}} \text{ and } \mathcal{S}(u, u') = \lambda \neq \mathbf{b} \} \\
& \cup \{ c(\bar{X}_n) \sim c'(\bar{Y}_n) \xleftarrow{\mathbf{t}} \text{pay}_{\lambda} \# ?, ((X_i \sim Y_i) \# ?)_{i=1 \dots n} \mid c, c' \in DC^n \\
& \text{and } \mathcal{S}(c, c') = \lambda \neq \mathbf{b} \} \\
& \cup \{ \text{pay}_{\lambda} \xleftarrow{\lambda} \mid \exists x, y \in S \text{ such that } \mathcal{S}(x, y) = \lambda \neq \mathbf{b} \}. \quad \square
\end{aligned}$$

The following lemma shows the relation between the semantics of equations in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ and the behaviour of the binary predicate symbol ' \sim ' defined by $EQ_{\mathcal{S}}$ in $\text{QCHL}(\mathcal{D}, \mathcal{C})$.

Lemma 4.1

Consider any two arbitrary terms t and s ; $EQ_{\mathcal{S}}$ defined as in Definition 4.1; and a satisfiable finite set Π of \mathcal{C} -constraints. Then, for every $d \in D \setminus \{\mathbf{b}\}$:

$$t \approx_{d, \Pi} s \iff EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$$

Proof

We separately prove each implication.

[\implies] Assume $t \approx_{d, \Pi} s$. Then, there are two terms \hat{t}, \hat{s} such that:

$$(1) t \approx_{\Pi} \hat{t} \quad (2) s \approx_{\Pi} \hat{s} \quad (3) \hat{t} \approx_d \hat{s}$$

We use structural induction on the form of the term \hat{t} .

- $\hat{t} = Z$, $Z \in \text{Var}$. From (3) we have $\hat{s} = Z$. Then (1) and (2) become $t \approx_{\Pi} Z$ and $s \approx_{\Pi} Z$, therefore $t \approx_{\Pi} s$. Now $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ can be proved with a proof tree rooted by a **QDA** step of the form:

$$\frac{(t == X\theta) \# \mathbf{t} \Leftarrow \Pi \quad (s == Y\theta) \# \mathbf{t} \Leftarrow \Pi \quad (X == Y)\theta \# \mathbf{t} \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

using the clause $X \sim Y \stackrel{\mathbf{t}}{\Leftarrow} (X == Y) \# ? \in EQ_{\mathcal{S}}$ instantiated by the substitution $\theta = \{X \mapsto t, Y \mapsto s\}$. Therefore the three premises can be derived from $EQ_{\mathcal{S}}$ with **QEA** steps since $t \approx_{\Pi} t$, $s \approx_{\Pi} s$ and $t \approx_{\Pi} s$, respectively. Checking the side conditions of all inference steps is straightforward.

- $\hat{t} = u$, $u \in B_{\mathcal{C}}$. From (3) we have $\hat{s} = u'$ for some $u' \in B_{\mathcal{C}}$ such that $d \trianglelefteq \lambda = \mathcal{S}(u, u')$. Then (1) and (2) become $t \approx_{\Pi} u$ and $s \approx_{\Pi} u'$, which allow to build a proof of $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ by means of a **QDA** step using the clause $u \sim u' \stackrel{\mathbf{t}}{\Leftarrow} \text{pay}_{\lambda} \# ?$.
- $\hat{t} = c$, $c \in DC^0$. From (3) we have $\hat{s} = c'$ for some $c' \in DC^0$ such that $d \trianglelefteq \lambda = \mathcal{S}(c, c')$. Then (1) and (2) become $t \approx_{\Pi} c$ and $s \approx_{\Pi} c'$, which allow us to build a proof of $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ by means of a **QDA** step using the clause $c \sim c' \stackrel{\mathbf{t}}{\Leftarrow} \text{pay}_{\lambda} \# ?$.
- $\hat{t} = c(\bar{t}_n)$, $c \in DC^n$ with $n > 0$. In this case, and because of (3), we can assume $\hat{s} = c'(\bar{s}_n)$ for some $c' \in DC^n$ satisfying $d \trianglelefteq d_0 =_{\text{def}} \mathcal{S}(c, c')$ and $d \trianglelefteq d_i =_{\text{def}} \mathcal{S}(t_i, s_i)$ for $i = 1 \dots n$. Then $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ with a proof tree rooted by a **QDA** step of the form:

$$\frac{\begin{array}{ll} (t == c(\bar{t}_n)) \# \mathbf{t} \Leftarrow \Pi & \text{pay}_{d_0} \# d_0 \Leftarrow \Pi \\ (s == c'(\bar{s}_n)) \# \mathbf{t} \Leftarrow \Pi & ((t_i \sim s_i) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \end{array}}{(t \sim s) \# d \Leftarrow \Pi}$$

using the $EQ_{\mathcal{S}}$ clause $C : c(\bar{X}_n) \sim c'(\bar{Y}_n) \stackrel{\mathbf{t}}{\Leftarrow} \text{pay}_{d_0} \# ?, ((X_i \sim Y_i) \# ?)_{i=1 \dots n}$ instantiated by the substitution $\theta = \{X_1 \mapsto t_1, Y_1 \mapsto s_1, \dots, X_n \mapsto t_n, Y_n \mapsto s_n\}$. Note that C has attenuation factor \mathbf{t} and threshold values $?$ at the body. Therefore, the

side conditions of the **QDA** step boil down to $d \trianglelefteq d_i$ ($1 \leq i \leq n$) which are true by assumption. It remains to prove that each premise of the **QDA** step can be derived from EQ_S in $\text{QCHL}(\mathcal{D}, \mathcal{C})$:

- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t == c(\bar{t}_n)) \# \mathbf{t} \Leftarrow \Pi$ and $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (s == c'(\bar{s}_n)) \# \mathbf{t} \Leftarrow \Pi$ are trivial consequences of $t \approx_{\Pi} c(\bar{t}_n)$ and $s \approx_{\Pi} c'(\bar{s}_n)$, respectively. In both cases, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs consist of one single **QEA** step.
- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} \text{pay}_{d_0} \# d_0 \Leftarrow \Pi$ can be proved using the clause $\text{pay}_{d_0} \xleftarrow{d_0} \in EQ_S$ in one single **QDA** step.
- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t_i \sim s_i) \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. For each i , we observe that $t_i \approx_{d_i, \Pi} s_i$ holds because of $\hat{t}_i = t_i$, $\hat{s}_i = s_i$ which satisfy $t_i \approx_{\Pi} \hat{t}_i$, $s_i \approx_{\Pi} \hat{s}_i$ and $\hat{t}_i \approx_{d_i} \hat{s}_i$. Since $\hat{t}_i = t_i$ is a subterm of $\hat{t} = c(\bar{t}_n)$, the inductive hypothesis can be applied.

[\Leftarrow] Let T be a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ -proof tree witnessing $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$. We prove $t \approx_{d, \Pi} s$ reasoning by induction on the number $n = \|T\|$ of nodes in T that represent conclusions of **QDA** inference steps. Note that all the program clauses belonging to EQ_S define either the binary predicate symbol ' \sim ' or the nullary predicates pay_{λ} .

Basis ($n = 1$).

In this case we have for the **QDA** inference step that there can be used three possible EQ_S clauses:

1. The program clause is $X \sim Y \xleftarrow{\mathbf{t}} (X == Y) \# ?$. Then the **QDA** inference step must be of the form:

$$\frac{(t == t') \# d_1 \Leftarrow \Pi \quad (s == s') \# d_2 \Leftarrow \Pi \quad (t' == s') \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. The proof of the three premises must use the **QEA** inference rule. Because of the conditions of this inference rule we have $t \approx_{\Pi} t'$, $s \approx_{\Pi} s'$ and $t' \approx_{\Pi} s'$. Therefore $t \approx_{\Pi} s$ is clear. Then $t \approx_{d, \Pi} s$ holds by taking $\hat{t} = \hat{s} = t$ because, trivially, $t \approx_{\Pi} \hat{t}$, $s \approx_{\Pi} \hat{s}$ and $\hat{t} \approx_d \hat{s}$.

2. The program clause is $u \sim u' \xleftarrow{\mathbf{t}} \text{pay}_{\lambda} \# ?$ with $u, u' \in B_C$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:

$$\frac{(t == u) \# d_1 \Leftarrow \Pi \quad (s == u') \# d_2 \Leftarrow \Pi \quad \text{pay}_{\lambda} \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the EQ_S clause $\text{pay}_{\lambda} \xleftarrow{\lambda}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteq \lambda$. Moreover, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_{\Pi} u$ and $s \approx_{\Pi} u'$. These facts and $u \approx_d u'$ imply $t \approx_{d, \Pi} s$.

3. The program clause is $c \sim c' \xleftarrow{\mathbf{t}} \text{pay}_{\lambda} \# ?$ with $c, c' \in DC^0$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:

$$\frac{(t == c) \# d_1 \Leftarrow \Pi \quad (s == c') \# d_2 \Leftarrow \Pi \quad \text{pay}_{\lambda} \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the EQ_S clause $\text{pay}_\lambda \xleftarrow{\lambda}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteq \lambda$. Moreover, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_\Pi c$ and $s \approx_\Pi c'$. These facts and $c \approx_d c'$ imply $t \approx_{d, \Pi} s$.

Inductive step ($n > 1$).

In this case t and s must be of the form $t = c(\bar{t}_n)$ and $s = c'(\bar{s}_n)$. The EQ_S clause used in the **QDA** inference step at the root must be of the form:

$$c(\bar{X}_n) \sim c'(\bar{Y}_n) \xleftarrow{\mathbf{t}} \text{pay}_{d_0} \#?, ((X_i \sim Y_i) \#?)_{i=1 \dots n}$$

with $\mathcal{S}(c, c') = d_0 \neq \mathbf{b}$. The inference step at the root will be:

$$\frac{\begin{array}{l} (t == c(\bar{t}_n)) \# d_1 \Leftarrow \Pi \quad \text{pay}_{d_0} \# e_0 \Leftarrow \Pi \\ (s == c'(\bar{s}_n)) \# d_2 \Leftarrow \Pi \quad ((t_i \sim s_i) \# e_i \Leftarrow \Pi)_{i=1 \dots n} \end{array}}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap \prod_{i=0}^n e_i$. Due to the forms of the EQ_S clause $\text{pay}_{d_0} \xleftarrow{d_0}$ and the **QEA** inference rule there is no loss of generality in assuming $d_1 = d_2 = \mathbf{t}$ and $e_0 = d_0$, therefore we have $d \trianglelefteq d_0 \sqcap \prod_{i=1}^n e_i$. By the inductive hypothesis $t_i \approx_{e_i, \Pi} s_i$ ($1 \leq i \leq n$), i.e. there are constructor terms \hat{t}_i, \hat{s}_i such that $t_i \approx_\Pi \hat{t}_i$, $s_i \approx_\Pi \hat{s}_i$ and $\hat{t}_i \approx_{e_i} \hat{s}_i$ for $i = 1 \dots n$. Thus, we can build $\hat{t} = c(\hat{t}_1, \dots, \hat{t}_n)$ and $\hat{s} = c'(\hat{s}_1, \dots, \hat{s}_n)$ having $t \approx_{d, \Pi} s$ because:

- $t \approx_\Pi \hat{t}$, i.e. $c(\bar{t}_n) \approx_\Pi c(\hat{t}_n)$, by decomposition since $t_i \approx_\Pi \hat{t}_i$.
- $s \approx_\Pi \hat{s}$, i.e. $c'(\bar{s}_n) \approx_\Pi c'(\hat{s}_n)$, again by decomposition since $s_i \approx_\Pi \hat{s}_i$.
- $\hat{t} \approx_d \hat{s}$, since $d \trianglelefteq d_0 \sqcap \prod_{i=1}^n e_i \trianglelefteq \mathcal{S}(c, c') \sqcap \prod_{i=1}^n \mathcal{S}(\hat{t}_i, \hat{s}_i) = \mathcal{S}(\hat{t}, \hat{s})$. \square

We are now ready to define elim_S acting over programs and goals.

Definition 4.2

Assume a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} and a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -goal G for \mathcal{P} whose atoms are all relevant for \mathcal{P} . Then we define:

1. For each atom A , let A_\sim be $t \sim s$ if $A : t == s$; otherwise let A_\sim be A .
2. For each clause $C : (p(\bar{t}_n) \xleftarrow{\alpha} \bar{B}) \in \mathcal{P}$ let $\hat{\mathcal{C}}_S$ be the set of $\text{QCLP}(\mathcal{D}, \mathcal{C})$ clauses consisting of:
 - The clause $\hat{C} : (\hat{p}_C(\bar{t}_n) \xleftarrow{\alpha} \bar{B}_\sim)$, where $\hat{p}_C \in DP^n$ is not affected by \mathcal{P} (chosen in a different way for each C) and \bar{B}_\sim is obtained from \bar{B} by replacing each atom A occurring in \bar{B} by A_\sim .
 - A clause $p'(\bar{X}_n) \xleftarrow{\mathbf{t}} \text{pay}_\lambda \#?, ((X_i \sim t_i) \#?)_{i=1 \dots n}$, $\hat{p}_C(\bar{t}_n) \#?$ for each $p' \in DP^n$ such that $\mathcal{S}(p, p') = \lambda \neq \mathbf{b}$. Here, \bar{X}_n must be chosen as n pairwise different variables not occurring in the clause C .
3. $\text{elim}_S(\mathcal{P})$ is the $\text{QCLP}(\mathcal{D}, \mathcal{C})$ -program $EQ_S \cup \hat{\mathcal{P}}_S$ where $\hat{\mathcal{P}}_S =_{\text{def}} \bigcup_{C \in \mathcal{P}} \hat{\mathcal{C}}_S$.
4. $\text{elim}_S(G)$ is the $\text{QCLP}(\mathcal{D}, \mathcal{C})$ -goal G_\sim obtained from G by replacing each atom A occurring in G by A_\sim . \square

The following example illustrates the transformation elim_S .

Example 4.2 (Running example: QCLP($\mathcal{U} \otimes \mathcal{W}$, \mathcal{R})-program $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$)

Consider the SQCLP(\mathcal{S}_r , $\mathcal{U} \otimes \mathcal{W}$, \mathcal{R})-program \mathcal{P}_r and the goal G_r for \mathcal{P}_r as presented in Example 4.1. The transformed QCLP($\mathcal{U} \otimes \mathcal{W}$, \mathcal{R})-program $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ is as follows:

$$\begin{array}{ll}
 \hat{r}_1 & \hat{famous}_{R_1}(sha) \xleftarrow{(0.9,1)} \\
 R_{1,1} & famous(X) \leftarrow pay_{\mathbf{t}}, X \sim sha, \hat{famous}_{R_1}(sha) \\
 \hat{r}_2 & \hat{wrote}_{R_2}(sha, kle) \xleftarrow{(1,1)} \\
 R_{2,1} & wrote(X, Y) \leftarrow pay_{\mathbf{t}}, X \sim sha, Y \sim kle, \hat{wrote}_{R_2}(sha, kle) \\
 R_{2,2} & authored(X, Y) \leftarrow pay_{(0.9,0)}, X \sim sha, Y \sim kle, \hat{wrote}_{R_2}(sha, kle) \\
 \hat{r}_3 & \hat{wrote}_{R_3}(sha, hamlet) \xleftarrow{(1,1)} \\
 R_{3,1} & wrote(X, Y) \leftarrow pay_{\mathbf{t}}, X \sim sha, Y \sim hamlet, \hat{wrote}_{R_3}(sha, hamlet) \\
 R_{3,2} & authored(X, Y) \leftarrow pay_{(0.9,0)}, X \sim sha, Y \sim hamlet, \hat{wrote}_{R_3}(sha, hamlet) \\
 \hat{r}_4 & \hat{good_work}_{R_4}(G) \xleftarrow{(0.75,3)} famous(A) \# (0.5, 100), authored(A, G) \\
 R_{4,1} & good_work(X) \leftarrow pay_{\mathbf{t}}, X \sim G, \hat{good_work}_{R_4}(G)
 \end{array}$$

% Program clauses for \sim :

$X \sim Y \leftarrow X == Y$

$kle \sim kli \leftarrow pay_{(0.8,2)}$

[...]

% Program clauses for pay:

$pay_{\mathbf{t}} \leftarrow$

$pay_{(0.9,0)} \xleftarrow{(0.9,0)}$

$pay_{(0.8,2)} \xleftarrow{(0.8,2)}$

Finally, the goal $\text{elim}_{\mathcal{S}}(G_r)$ for $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ is as follows:

$$good_work(X) \# W \parallel W \triangleright^? (0.5, 10) \quad \square$$

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.1

Consider a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , an atom A relevant for \mathcal{P} , a qualification value $d \in D \setminus \{\mathbf{b}\}$ and a satisfiable finite set of \mathcal{C} -constraints Π . Then, the following two statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$
2. $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$

where A_{\sim} is understood as in Definition 4.2(1).

Proof

We separately prove each implication.

[1. \Rightarrow 2.] (*the transformation is complete*). Assume that T is a SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) proof tree witnessing $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We want to show the existence of a QCHL(\mathcal{D}, \mathcal{C}) proof tree T' witnessing $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$. We reason by complete induction on $\|T\|$. There are three possible cases according to the syntactic form of the atom A . In each case we argue how to build the desired proof tree T' .
— A is a primitive atom κ . In this case A_{\sim} is also κ and T contains only one **SQPA** inference node. Because of the inference rules **SQPA** and **QPA**, both

$\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ and $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ are equivalent to $\Pi \models_{\mathcal{C}} \kappa$, therefore T' trivially contains just one **QPA** inference node.

— A is an equation $t = s$. In this case A_{\sim} is $t \sim s$ and T contains just one **SQEA** inference node. We know $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t = s) \# d \Leftarrow \Pi$ is equivalent to $t \approx_{d, \Pi} s$ because of the inference rule **SQEA**. From this equivalence follows $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ due to Lemma 4.1 and hence $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ by construction of $\text{elim}_{\mathcal{S}}(\mathcal{P})$. In this case, T' will be a proof tree rooted by a **QDA** inference step.

— A is a defined atom $p'(\bar{t}'_n)$ with $p' \in DP^n$. In this case A_{\sim} is $p'(\bar{t}'_n)$ and the root inference of T must be a **SQDA** inference step of the form:

$$\frac{(\ (t'_i = t_i \theta) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi \)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\clubsuit)$$

with $C : (p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}$, θ substitution, $\mathcal{S}(p', p) = d_0 \neq \mathbf{b}$, $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \trianglelefteq d_i$ ($0 \leq i \leq n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \leq j \leq m$)—which means $d \trianglelefteq \alpha$ in the case $m = 0$. We can assume that the first n premises at (\clubsuit) are proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} by proof trees T_{1i} ($1 \leq i \leq n$) satisfying $\|T_{1i}\| < \|T\|$ ($1 \leq i \leq n$), and the last m premises at (\clubsuit) are proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} by proof trees T_{2j} ($1 \leq j \leq m$) satisfying $\|T_{2j}\| < \|T\|$ ($1 \leq j \leq m$).

By Definition 4.2, we know that the transformed program $\text{elim}_{\mathcal{S}}(\mathcal{P})$ contains two clauses of the following form:

$$\begin{aligned} \hat{C} : \quad & \hat{p}_C(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m \\ \hat{C}_{p'} : \quad & p'(\bar{X}_n) \xleftarrow{t} \text{pay}_{d_0} \# d_0?, \ (X_i \sim t_i) \# d_i? \)_{i=1 \dots n}, \ \hat{p}_C(\bar{t}_n) \# d_0? \end{aligned}$$

where X_i ($1 \leq i \leq n$) are fresh variables not occurring in C and B_j ($1 \leq j \leq m$) is the result of replacing ' \sim ' for ' $=$ ' if B_j is equation; and B_j itself otherwise. Given that the n variables X_i do not occur in C , we can assume that $\sigma =_{\text{def}} \theta' \uplus \theta$ with $\theta' =_{\text{def}} \{X_1 \mapsto t'_1, \dots, X_n \mapsto t'_n\}$ is a well-defined substitution. We claim that $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$ can be proved with a proof tree T' rooted by the **QDA** inference step $(\spadesuit.1)$, which uses the clause $\hat{C}_{p'}$ instantiated by σ and having $d_{n+1} = d$.

$$\frac{\begin{array}{l} (\ (t'_i = X_i \sigma) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \text{pay}_{d_0} \sigma \# d_0 \Leftarrow \Pi \\ (\ (X_i \sim t_i) \sigma \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n) \sigma \# d_{n+1} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit.1) \quad \frac{\begin{array}{l} (\ (t'_i = X_i \theta') \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \text{pay}_{d_0} \# d_0 \Leftarrow \Pi \\ (\ (X_i \theta' \sim t_i \theta) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n \theta) \# d_{n+1} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit.2)$$

By construction of σ , $(\spadesuit.1)$ can be rewritten as $(\spadesuit.2)$, and in order to build the rest of T' , we show that each premise of $(\spadesuit.2)$ admits a proof in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ w.r.t. the transformed program $\text{elim}_{\mathcal{S}}(\mathcal{P})$:

- $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t'_i = X_i \theta') \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. Straightforward using a single **QEA** inference step since $X_i \theta' = t'_i$ and $t'_i \approx_{\Pi} t'_i$ is trivially true.
- $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \text{pay}_{d_0} \# d_0 \Leftarrow \Pi$. Immediate using the clause $(\text{pay}_{d_0} \xleftarrow{d_0}) \in \text{elim}_{\mathcal{S}}(\mathcal{P})$ with a single **QDA** inference step.
- $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (X_i \theta' \sim t_i \theta) \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. From the first n premises of (\clubsuit)

we know $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t'_i == t_i \theta) \sharp d_i \Leftarrow \Pi$ with a proof tree T_{1i} satisfying $\|T_{1i}\| < \|T\|$ for $i = 1 \dots n$. Therefore, for $i = 1 \dots n$, $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t'_i \sim t_i \theta) \sharp d_i \Leftarrow \Pi$ with some $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T'_{1i} by inductive hypothesis. Since $(X_i \theta' \sim t_i \theta) = (t'_i \sim t_i \theta)$ for $i = 1 \dots n$, we are done.

- $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \hat{p}_C(\bar{t}_n \theta) \sharp d \Leftarrow \Pi$. This is proved by a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree with a **QDA** inference step node at its root of the following form:

$$\frac{(\ (t_i \theta == t_i \theta) \sharp d_i \Leftarrow \Pi \)_{i=1 \dots n} \quad (\ B_j \theta \sharp e_j \Leftarrow \Pi \)_{j=1 \dots m}}{\hat{p}_C(\bar{t}_n \theta) \sharp d \Leftarrow \Pi} \quad (\heartsuit)$$

which uses the program clause \hat{C} instantiated by the substitution θ . Once more, we have to check that the premises can be derived in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ from the transformed program $\text{elim}_{\mathcal{S}}(\mathcal{P})$ and that the side conditions of (\heartsuit) are satisfied:

- The first n premises can be trivially proved using **QEA** inference steps.
- The last m premises can be proved w.r.t. $\text{elim}_{\mathcal{S}}(\mathcal{P})$ with some $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof trees T'_{2j} ($1 \leq j \leq m$) by the inductive hypothesis, since we have premises $(\ B_j \theta \sharp e_j \Leftarrow \Pi \)_{j=1 \dots m}$ at (\clubsuit) that can be proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} with proof trees T_{2j} of size $\|T_{2j}\| < \|T\|$ ($1 \leq j \leq m$).
- The side conditions—namely: $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \trianglelefteq d_i$ ($1 \leq i \leq n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \leq j \leq m$)—trivially hold because they are also satisfied by (\clubsuit) .

Finally, we complete the construction of T' by checking that $(\spadesuit.2)$ satisfies the side conditions of the inference rule **QDA**:

- All threshold values at the body of $\hat{C}_{p'}$ are '?', therefore the first group of side conditions becomes $d_i \triangleright^? ?$ ($0 \leq i \leq n+1$), which are trivially true.
- The second side condition reduces to $d \trianglelefteq \mathbf{t}$, which is also trivially true.
- The third, and last, side condition is $d \trianglelefteq \mathbf{t} \circ d_i$ ($0 \leq i \leq n+1$), or equivalently $d \trianglelefteq d_i$ ($0 \leq i \leq n+1$). In fact, $d \trianglelefteq d_i$ ($0 \leq i \leq n$) holds due to the side conditions in (\clubsuit) , and $d \trianglelefteq d_{n+1}$ holds because $d_{n+1} = d$ by construction of $(\spadesuit.1)$ and $(\spadesuit.2)$.

[2. \Rightarrow 1.] (*the transformation is sound*). Assume that T' is a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree witnessing $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \sharp d \Leftarrow \Pi$. We want to show the existence of a $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \sharp d \Leftarrow \Pi$. We reason by complete induction of $\|T'\|$. There are three possible cases according to the syntactic form of the atom A_{\sim} . In each case we argue how to build the desired proof tree T .

— A_{\sim} is a primitive atom κ . In this case A is also κ and T' contains only one **QPA** inference node. Both $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \kappa \sharp d \Leftarrow \Pi$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \kappa \sharp d \Leftarrow \Pi$ are equivalent to $\Pi \models_{\mathcal{C}} \kappa$ because of the inference rules **QPA** and **SQPA**, therefore T trivially contains just one **SQPA** inference node.

— A_{\sim} is of the form $t \sim s$. In this case A is $t == s$ and T' is rooted by a **QDA** inference step. From $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \sharp d \Leftarrow \Pi$ and by construction of $\text{elim}_{\mathcal{S}}(\mathcal{P})$ we have $\text{EQS} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \sharp d \Leftarrow \Pi$. By Lemma 4.1 we get $t \approx_{d, \Pi} s$ and, by the definition of the **SQEA** inference step, we can build T as a proof tree with only one **SQEA** inference node proving $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t == s) \sharp d \Leftarrow \Pi$.

— A_{\sim} is a defined atom $p'(\bar{t}_n)$ with $p' \in DP^n$ and $p' \neq \sim$. In this case $A = A_{\sim}$ and the step at the root of T' must be a **QDA** inference step using a clause $C' \in$

$\text{elim}_S(\mathcal{P})$ with head predicate p' and a substitution θ . Because of Definition 4.2 and the fact that p' is relevant for \mathcal{P} , there must be some clause $C : (p(\bar{t}_n) \leftarrow^\alpha \bar{B}) \in \mathcal{P}$ such that $S(p, p') = d_0 \neq \mathbf{b}$, and C' must be of the form:

$$C' : p'(\bar{X}_n) \leftarrow^t \text{pay}_{d_0} \#?, ((X_i \sim t_i) \#?)_{i=1 \dots n}, \hat{p}_C(\bar{t}_n) \#?$$

where the variables \bar{X}_n do not occur in C . Thus the **QDA** inference step at the root of T' must be of the form:

$$\frac{\begin{array}{l} ((t'_i == X_i \theta) \# d_{1i} \Leftarrow \Pi)_{i=1 \dots n} \\ \text{pay}_{d_0} \theta \# e_{10} \Leftarrow \Pi \\ ((X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n) \theta \# e_{1(n+1)} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit)$$

and the proof of the last premise must use the only clause for \hat{p}_C introduced in $\text{elim}_S(\mathcal{P})$ according to Definition 4.2, i.e.:

$$\hat{C} : \hat{p}_C(\bar{t}_n) \leftarrow^\alpha B_{\sim}^1 \# w_1, \dots, B_{\sim}^m \# w_m.$$

Therefore, the proof of this premise must be of the form:

$$\frac{((t_i \theta == t_i \theta') \# d_{2i} \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j^j \theta' \# e_{2j} \Leftarrow \Pi)_{j=1 \dots m}}{\hat{p}_C(\bar{t}_n) \theta \# e_{1(n+1)} \Leftarrow \Pi} \quad (\heartsuit)$$

for some substitution θ' not affecting \bar{X}_n . We can assume that the last m premises in (\heartsuit) are proved in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ w.r.t. $\text{elim}_S(\mathcal{P})$ by proof trees T'_j satisfying $\|T'_j\| < \|T'\|$ ($1 \leq j \leq m$). Then we use the substitution θ' and clause C to build a $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree T with a **SQDA** inference step at the root of the form:

$$\frac{((t'_i == t_i \theta') \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta' \# e_{2j} \Leftarrow \Pi)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\clubsuit)$$

Next we check that the premises of this inference step admit proofs in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ and that (\clubsuit) satisfies the side conditions of a valid **SQDA** inference step.

- $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t'_i == t_i \theta') \# e_{1i} \Leftarrow \Pi$ for $i = 1 \dots n$.
 - From the premises $((X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ of (\spadesuit) and by construction of $\text{elim}_S(\mathcal{P})$ we know $EQ_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi$ ($1 \leq i \leq n$). Therefore by Lemma 4.1 we have $X_i \theta \approx_{e_{1i}, \Pi} t_i \theta$ for $i = 1 \dots n$.
 - Consider now the premises $((t'_i == X_i \theta) \# d_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ of (\spadesuit) . Their proofs must rely on **QEA** inference steps, and therefore $t'_i \approx_{\Pi} X_i \theta$ holds for $i = 1 \dots n$.
 - Analogously, from the proofs of the premises $((t_i \theta == t_i \theta') \# d_{2i} \Leftarrow \Pi)_{i=1 \dots n}$ we have $t_i \theta \approx_{\Pi} t_i \theta'$ (or equivalently $t_i \theta' \approx_{\Pi} t_i \theta$) for $i = 1 \dots n$.

From the previous points we have $X_i \theta \approx_{e_{1i}, \Pi} t_i \theta$, $t'_i \approx_{\Pi} X_i \theta$ and $t_i \theta' \approx_{\Pi} t_i \theta$, which by Lemma 2.7(1) of (Rodríguez-Artalejo and Romero-Díaz 2010b) imply $t'_i \approx_{e_{1i}, \Pi} t_i \theta'$ ($1 \leq i \leq n$). Therefore the premises $((t'_i == t_i \theta') \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ can be proven in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ using a **SQEA** inference step.

- $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} B_j \theta' \# e_{2j} \Leftarrow \Pi$ for $j = 1 \dots m$. We know $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} B_j^i \theta' \# e_{2j} \Leftarrow \Pi$ with a proof tree T_j' satisfying $\|T_j'\| < \|T'\|$ ($1 \leq j \leq m$) because of (\heartsuit). Therefore we have, by inductive hypothesis, $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} B_j \theta' \# e_{2j} \Leftarrow \Pi$ for some SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) proof tree T_j ($1 \leq j \leq m$).
- $\mathcal{S}(p, p') = d_0 \neq \mathbf{b}$. As seen above.
- $e_{2j} \triangleright^? w_j$ for $j = 1 \dots m$. This is a side condition of the **QDA** step in (\heartsuit).
- $d \trianglelefteq e_{1i}$ for $i = 1 \dots n$. Straightforward from the side conditions of (\spadesuit), which include $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $(0 \leq i \leq n+1)$.
- $d \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \dots m$. This follows from the side conditions of (\spadesuit) and (\heartsuit), since we have $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $i = 0 \dots n+1$ (in particular $d \trianglelefteq e_{1(n+1)}$) and $e_{1(n+1)} \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \dots m$. \square

Finally, the next theorem extends the previous result to goals.

Theorem 4.2

Let G be a goal for a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} whose atoms are all relevant for \mathcal{P} . Assume $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ and $G' = \text{elim}_{\mathcal{S}}(G)$. Then, $\text{Sol}_{\mathcal{P}}(G) = \text{Sol}_{\mathcal{P}'}(G')$.

Proof

According to the definition of goals in Section 2, and Definition 4.2, G and G' must be of the form $(A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$ and $(A_i^i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$, respectively. By Definitions 2.2 and 3.1, both $\text{Sol}_{\mathcal{P}}(G)$ and $\text{Sol}_{\mathcal{P}'}(G')$ are sets of triples $\langle \sigma, \mu, \Pi \rangle$ where σ is a \mathcal{C} -substitution, $\mu : \text{war}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ (note that $\text{war}(G) = \text{war}(G')$) and Π is a satisfiable finite set of \mathcal{C} -constraints. Moreover:

1. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ iff $W_i \mu = d_i \triangleright^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$ ($1 \leq i \leq m$).
2. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ iff $W_i \mu = d_i \triangleright^? \beta_i$ and $\mathcal{P}' \vdash_{\mathcal{D}, \mathcal{C}} A_i^i \sigma \# W_i \mu \Leftarrow \Pi$ ($1 \leq i \leq m$).

Because of Theorem 4.1, conditions (1) and (2) are equivalent. \square

4.2 Transforming QCLP into CLP

The results presented in this subsection are dependant on the assumption that the qualification domain \mathcal{D} is existentially expressible in the constraint domain \mathcal{C} via an injective mapping $\iota : D_{\mathcal{D}} \setminus \{\mathbf{b}\} \rightarrow C_{\mathcal{C}}$ and two existential \mathcal{C} -constraints of the following form:

$$\begin{aligned} \text{qVal}(X) &= \exists U_1 \dots \exists U_k (B_1 \wedge \dots \wedge B_m) \\ \text{qBound}(X, Y, Z) &= \exists V_1 \dots \exists V_l (C_1 \wedge \dots \wedge C_q) \end{aligned}$$

The intuition behind $\text{qVal}(X)$ and $\text{qBound}(X, Y, Z)$ has been explained in Definition 2.1. Roughly, they are intended to represent qualification values from \mathcal{D} and the behaviour of \mathcal{D} 's attenuation operator \circ by means of \mathcal{C} -constraints. Moreover, the assumption that $\text{qVal}(X)$ and $\text{qBound}(X, Y, Z)$ have the existential form displayed above allows to build CLP clauses for two predicate symbols $qVal \in DP^1$ and $qBound \in DP^3$ which will capture the behaviour of the two corresponding constraints in the sense of Lemma 3.1. More precisely, we consider the CLP(\mathcal{C})-program $E_{\mathcal{D}}$ consisting of the following two clauses:

$$\begin{aligned} qVal(X) &\leftarrow B_1, \dots, B_m \\ qBound(X, Y, Z) &\leftarrow C_1, \dots, C_q \end{aligned}$$

The next example shows the CLP clauses in $E_{\mathcal{D}}$ for $\mathcal{C} = \mathcal{R}$ and three different choices of a qualification domain \mathcal{D} that is existentially expressible in \mathcal{R} , namely: \mathcal{U} , \mathcal{W} and $\mathcal{U} \otimes \mathcal{W}$. In each case, the CLP clauses in $E_{\mathcal{D}}$ are obtained straightforwardly from the \mathcal{R} constraints $\mathbf{qVal}(X)$ and $\mathbf{qBound}(X, Y, Z)$ shown in Example 2.1.

Example 4.3

1. $E_{\mathcal{U}}$ consists of the following two clauses:

$$\begin{aligned} qVal(X) &\leftarrow cp_{<}(0, X), \quad cp_{\leq}(X, 1) \\ qBound(X, Y, Z) &\leftarrow op_{\times}(Y, Z, X'), \quad cp_{\leq}(X, X') \end{aligned}$$

2. $E_{\mathcal{W}}$ consists of the following two clauses:

$$\begin{aligned} qVal(X) &\leftarrow cp_{\geq}(X, 0) \\ qBound(X, Y, Z) &\leftarrow op_{+}(Y, Z, X'), \quad cp_{\geq}(X, X') \end{aligned}$$

3. $E_{\mathcal{U} \otimes \mathcal{W}}$ consists of the following two clauses:

$$\begin{aligned} qVal(X) &\leftarrow X == \mathbf{pair}(X_1, X_2), \quad cp_{<}(0, X_1), \quad cp_{\leq}(X_1, 1), \quad cp_{\geq}(X_2, 0) \\ qBound(X, Y, Z) &\leftarrow X == \mathbf{pair}(X_1, X_2), \quad Y == \mathbf{pair}(Y_1, Y_2), \quad Z == \mathbf{pair}(Z_1, Z_2), \\ &\quad op_{\times}(Y_1, Z_1, X'_1), \quad cp_{\leq}(X_1, X'_1), \quad op_{+}(Y_2, Z_2, X'_2), \quad cp_{\geq}(X_2, X'_2) \quad \square \end{aligned}$$

In general, the CLP clauses in $E_{\mathcal{D}}$ along with other techniques explained in the rest of this subsection will be used to present semantically correct transformations from $\text{QCLP}(\mathcal{D}, \mathcal{C})$ into $\text{CLP}(\mathcal{C})$, working both for programs and goals. All our results will work under the assumption that $qVal \in DP^1$ and $qBound \in DP^3$ are chosen as fresh predicate symbols not occurring in the $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals to be transformed. The next technical lemma ensures that the predicates $qVal$ and $qBound$ correctly represent the behaviour of the constraints $\mathbf{qVal}(X)$ and $\mathbf{qBound}(X, Y, Z)$.

Lemma 4.2

For any satisfiable finite set Π of \mathcal{C} -constraints one has:

1. For any ground term $t \in C_{\mathcal{C}}$:

$$t \in \text{ran}(\iota) \iff \mathbf{qVal}(t) \text{ true in } \mathcal{C} \iff E_{\mathcal{D}} \vdash_{\mathcal{C}} qVal(t) \Leftarrow \Pi$$

2. For any ground terms $r = \iota(x)$, $s = \iota(y)$, $t = \iota(z)$ with $x, y, z \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$:

$$x \trianglelefteq y \circ z \iff \mathbf{qBound}(r, s, t) \text{ true in } \mathcal{C} \iff E_{\mathcal{D}} \vdash_{\mathcal{C}} qBound(r, s, t) \Leftarrow \Pi$$

The two items above are also valid if $E_{\mathcal{D}}$ is replaced by any $\text{CLP}(\mathcal{C})$ -program including the two clauses in $E_{\mathcal{D}}$ and having no additional occurrences of $qVal$ and $qBound$ at the head of clauses.

Proof

Immediate consequence of Lemma 3.1 and Definition 2.1. \square

Now we are ready to define the transformations from $\text{QCLP}(\mathcal{D}, \mathcal{C})$ into $\text{CLP}(\mathcal{C})$.

Transforming Atoms

$$\mathbf{TEA} \quad (t == s)^\mathcal{T} = (t == s, \iota(\mathbf{t})).$$

$$\mathbf{TPA} \quad (\kappa)^\mathcal{T} = (\kappa, \iota(\mathbf{t})) \text{ with } \kappa \text{ primitive atom.}$$

$$\mathbf{TDA} \quad (p(\bar{t}_n))^\mathcal{T} = (p'(\bar{t}_n, W), W) \text{ with } p \in DP^n \text{ and } W \text{ a fresh CLP variable.}$$

Transforming qc-Atoms

$$\mathbf{TQCA} \quad \frac{A^\mathcal{T} = (A', w)}{(A \# d \Leftarrow \Pi)^\mathcal{T} = (A' \Leftarrow \Pi, \{qVal(w), qBound(\iota(d), \iota(\mathbf{t}), w)\})}$$

Transforming Program Clauses

$$\mathbf{TPC} \quad \frac{(\ B_j^\mathcal{T} = (B'_j, w'_j) \)_{j=1\dots m}}{C^\mathcal{T} = p'(\bar{t}_n, W) \leftarrow qVal(W), \left(\begin{array}{l} qVal(w'_j), \lceil w'_j \rceil \triangleright^? \iota(w_j)^\neg, \\ qBound(W, \iota(\alpha), w'_j), B'_j \end{array} \right)_{j=1\dots m}}$$

where $C : p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m$, W is a fresh CLP variable and $\lceil w'_j \rceil \triangleright^? \iota(w_j)^\neg$ is omitted if $w_j = ?$, otherwise abbreviates $qBound(\iota(w_j), \iota(\mathbf{t}), w'_j)$.

Transforming Goals

$$\mathbf{TG} \quad \frac{(\ B_j^\mathcal{T} = (B'_j, w'_j) \)_{j=1\dots m}}{\text{elim}_{\mathcal{D}}(G) = \left(\begin{array}{l} qVal(W_j), \lceil W_j \rceil \triangleright^? \iota(\beta_j)^\neg, \\ qVal(w'_j), qBound(W_j, \iota(\mathbf{t}), w'_j), B'_j \end{array} \right)_{j=1\dots m}}$$

where $G : (B_j \# W_j, W_j \triangleright^? \beta_j)_{j=1\dots m}$ and $\lceil W_j \rceil \triangleright^? \iota(\beta_j)^\neg$ as in **TPC** above.

Fig. 5. Transformation rules

Definition 4.3

Assume that \mathcal{D} is existentially expressible in \mathcal{C} , and let $qVal(X)$, $qBound(X, Y, Z)$ and $E_{\mathcal{D}}$ be as explained above. Assume also a QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P} and a QCLP(\mathcal{D}, \mathcal{C})-goal G for \mathcal{P} without occurrences of the defined predicate symbols $qVal$ and $qBound$. Then:

1. \mathcal{P} is transformed into the CLP(\mathcal{C})-program $\text{elim}_{\mathcal{D}}(\mathcal{P})$ consisting of the two clauses in $E_{\mathcal{D}}$ and the transformed $C^\mathcal{T}$ of each clause $C \in \mathcal{P}$, built as specified in Figure 5. The transformation rules of this figure translate each n -ary predicate symbol $p \in DP^n$ into a different $(n+1)$ -ary predicate symbol $p' \in DP^{n+1}$.
2. G is transformed into the CLP(\mathcal{C})-goal $\text{elim}_{\mathcal{D}}(G)$ built as specified in Figure 5. Note that the qualification variables \bar{W}_n occurring in G become normal CLP variables in the transformed goal. \square

The first three rules in Figure 5 are used for transforming atoms. For convenience, the transformation of an atom produces a pair where the first value is the transformed atom and the second one is either a new variable or the representation

of \mathbf{t} . In the first two cases, namely **TEA** and **TPA**, the transformation behaves as the identity and no new variables are introduced. The third case, namely **TDA**, corresponds to the transformation of a defined atom. In this case, a new CLP variable W —intended to represent the qualification value associated to the atom—is added as its last argument. The rule **TQCA** transforms qc-atoms of the form $A\#d \leftarrow \Pi$ by means of the transformation of A using one of the three aforementioned transformation rules. This transformation returns a pair (A', w) in which, as shown above, w can be either a new variable or the representation of \mathbf{t} . Since w can be a new variable W , the constraint $\mathbf{qVal}(w)$ is introduced to ensure that it represents a qualification value. Finally, the constraint $\mathbf{qBound}(\iota(d), \iota(\mathbf{t}), w)$ encodes “ $d \leq \mathbf{t} \circ w$,” or equivalently “ $d \geq w$.” The rule **TPC** is employed for transforming program clauses $C : p(\bar{t}_n) \leftarrow^\alpha B_1\#w_1, \dots, B_m\#w_m$ where each w_i is either a qualification value or $?$ indicating that proving the atom with any qualification value different from \mathbf{b} is acceptable. The rule introduces a new variable W together with a constraint $\mathbf{qVal}(W)$. The variable represents the qualification value associated to the computation of user defined atoms involving p (renamed as p' in the transformed program). The premises $(B_j^T = (B'_j, w'_j))_{j=1\dots m}$ transform the atoms in the body of the clause using in each case either **TEA**, **TPA** or **TDA**. Therefore, each w'_j obtained in this way represents a qualification value encoded as a constraint value. Moreover, the qualification value encoded by w'_j must be greater or equal than the corresponding qualification value w_j that occurs in the program clause. These two requirements are represented as $\mathbf{qVal}(w'_j), \lceil w'_j \geq^? \iota(w_j) \rceil$ in the transformed clause. The predicate call $\mathbf{qBound}(W, \iota(\alpha), w'_j)$ ensures that the value in W must be less than or equal to “ $\alpha \circ w'_j$ ” for every j . For each $j = 1 \dots m$ all the atoms associated to the transformation of B_j precede the transformed atom B'_j . In a Prolog-based implementation, this helps to prune the search space as soon as possible during the computations. The ideas behind rule **TG** are similar. A goal $G : (B_j\#W_j, W_j \geq^? \beta_j)_{j=1\dots m}$ is transformed by introducing atoms in charge of checking that: each W_j is a valid qualification value; each W_j is indeed less than or equal to the representation of β_j in CLP; each value w_j —obtained during the transformation of the atoms B_j —corresponds to an actual qualification value; and finally, that each W_j is satisfactory—i.e. less or equal to—w.r.t. its corresponding w_j before effectively introducing the transformed atoms B'_j . The following example illustrates the transformation $\text{elim}_{\mathcal{D}}$.

Example 4.4 (Running example: $\text{CLP}(\mathcal{R})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$)

Consider the $\text{QCLP}(\mathcal{U} \otimes \mathcal{W}, \mathcal{R})$ -program $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ and the goal $\text{elim}_{\mathcal{S}}(G_r)$ for the same program as presented in Example 4.2. The transformed $\text{CLP}(\mathcal{R})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ is as follows:

$$\begin{aligned}
 \hat{R}_1 \quad & \hat{famous}_{R_1}(sha, W) \leftarrow \mathbf{qVal}(W), \mathbf{qBound}(W, \mathbf{t}, (0.9, 1)) \\
 R_{1.1} \quad & famous(X, W) \leftarrow \mathbf{qVal}(W), \mathbf{qVal}(W_1), \mathbf{qBound}(W, \mathbf{t}, W_1), \text{payt}(W_1), \\
 & \mathbf{qVal}(W_2), \mathbf{qBound}(W, \mathbf{t}, W_2), \sim(X, sha, W_2), \\
 & \mathbf{qVal}(W_3), \mathbf{qBound}(W, \mathbf{t}, W_3), \hat{famous}_{R_1}(sha, W_3)
 \end{aligned}$$

```

 $\hat{R}_2$     $\hat{wrote}_{R_2}(sha, kle, W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (1,1))$ 
 $R_{2,1}$   $wrote(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1),$ 
       $qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2),$ 
       $qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, kle, W_3),$ 
       $qVal(W_4), qBound(W, \mathbf{t}, W_4), \hat{wrote}_{R_2}(sha, kle, W_4)$ 
 $R_{2,2}$   $authored(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{(0.9,0)}(W_1),$ 
       $qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2),$ 
       $qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, kle, W_3),$ 
       $qVal(W_4), qBound(W, \mathbf{t}, W_4), \hat{wrote}_{R_2}(sha, kle, W_4)$ 
 $\hat{R}_3$     $\hat{wrote}_{R_3}(sha, hamlet, W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (1,1))$ 
 $R_{3,1}$   $wrote(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1),$ 
       $qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2),$ 
       $qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, hamlet, W_3),$ 
       $qVal(W_4), qBound(W, \mathbf{t}, W_4), \hat{wrote}_{R_3}(sha, hamlet, W_4)$ 
 $R_{3,2}$   $authored(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{(0.9,0)}(W_1),$ 
       $qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2),$ 
       $qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, hamlet, W_3),$ 
       $qVal(W_4), qBound(W, \mathbf{t}, W_4), \hat{wrote}_{R_3}(sha, hamlet, W_4)$ 
 $\hat{R}_4$     $\hat{good\_work}_{R_4}(G, W) \leftarrow qVal(W),$ 
       $qVal(W_1), qBound((0.5,100), \mathbf{t}, W_1), qBound(W, (0.75,3), W_1), famous(A, W_1),$ 
       $qVal(W_2), qBound(W, (0.75,3), W_2), authored(A, G, W_2)$ 
 $R_{4,1}$   $good\_work(X, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1),$ 
       $qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, G, W_2),$ 
       $qVal(W_3), qBound(W, \mathbf{t}, W_3), \hat{good\_work}_{R_4}(G, W_3)$ 

```

% Program clauses for \sim :

```

 $\sim(X, Y, W) \leftarrow qVal(W), qVal(\mathbf{t}), qBound(W, \mathbf{t}, \mathbf{t}), X=Y$ 
 $\sim(kle, kli, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{(0.8,2)}(W_1)$ 
[...]
```

% Program clauses for pay :

```

 $pay_{\mathbf{t}}(W) \leftarrow qVal(W), qBound(W, \mathbf{t}, \mathbf{t})$ 
 $pay_{(0.9,0)}(W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (0.9,0))$ 
 $pay_{(0.8,2)}(W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (0.8,2))$ 

```

% Program clauses for $qVal$ & $qBound$:

```

 $qVal((X_1, X_2)) \leftarrow X_1 > 0, X_1 \leq 1, X_2 \geq 0$ 
 $qBound((W_1, W_2), (Y_1, Y_2), (Z_1, Z_2)) \leftarrow W_1 \leq Y_1 \times Z_1, W_2 \geq Y_2 + Z_2$ 

```

Finally, the goal $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G_r))$ for $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ is as follows:

```

 $qVal(W), qBound((0.5,10), \mathbf{t}, W), qVal(W'), qBound(W, \mathbf{t}, W'), good\_work(X, W')$ 

```

Note that, in order to improve the clarity of the program clauses of this example, the qualification value (1,0)—top value in $\mathcal{U} \otimes \mathcal{W}$ —has been replaced by \mathbf{t} . \square

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.3

Let A be an atom such that $qVal$ and $qBound$ do not occur in A . Assume $d \in D \setminus \{\mathbf{b}\}$ such that $(A\#d \Leftarrow \Pi)^T = (A' \Leftarrow \Pi, \Omega)$. Then, the two following statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A\#d \Leftarrow \Pi$
2. $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$.

Proof

We separately prove each implication.

[1. \Rightarrow 2.] (*the transformation is complete*). We assume that T is a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree witnessing $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A\#d \Leftarrow \Pi$. We want to show the existence of a $\text{CLP}(\mathcal{C})$ proof tree T' witnessing $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$. We reason by complete induction on $\|T\|$. There are three possible cases, according to the syntactic form of the atom A . In each case we argue how to build the desired proof tree T' .

— A is a primitive atom κ . In this case **TQCA** and **TPA** compute $A' = \kappa$ and $\Omega = \{\mathbf{qVal}(\iota(\mathbf{t})), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \kappa\#d \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **QPA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa\varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **PA** node. Moreover, $\varepsilon \in \text{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in \mathcal{C} .

— A is an equation $t == s$. In this case **TQCA** and **TEA** compute $A' = (t == s)$ and $\Omega = \{\mathbf{qVal}(\iota(\mathbf{t})), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t == s)\#d \Leftarrow \Pi$ follows $t \approx_{\Pi} s$ due to the **QEA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s)\varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **EA** node. Moreover, $\varepsilon \in \text{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in \mathcal{C} .

— A is a defined atom $p(\overline{t}_n)$ with $p \in DP^n$. In this case **TQCA** and **TDA** compute $A' = p'(\overline{t}_n, W)$ and $\Omega = \{\mathbf{qVal}(W), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), W)\}$ where W is a fresh CLP variable. On the other hand, T must be rooted by a **QDA** step of the form:

$$\frac{(\iota'_i == \iota_i \theta)\#d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p(\overline{t}_n)\#d \Leftarrow \Pi} \quad (\clubsuit)$$

using a clause $C : (p(\overline{t}_n) \Leftarrow^{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}$ instantiated by a substitution θ and such that the side conditions $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \trianglelefteq d_i$ ($1 \leq i \leq n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \leq j \leq m$) are fulfilled.

For $j = 1 \dots m$ we can assume $B_j^T = (B'_j, w'_j)$ and thus $(B_j \theta \# e_j \Leftarrow \Pi)^T = (B'_j \theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\mathbf{qVal}(w'_j), \mathbf{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j)\}$. The proof trees T_j of the last m premises of (\clubsuit) will have less than $\|T\|$ nodes, and hence the induction hypothesis can be applied to each $(B_j \theta \# e_j \Leftarrow \Pi)$ with $1 \leq j \leq m$, obtaining $\text{CHL}(\mathcal{C})$ proof trees T'_j proving $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta \rho_j \Leftarrow \Pi$ for some $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ with $\text{dom}(\rho_j) = \text{var}(\Omega_j)$.

Consider $\rho = \{W \mapsto \iota(d)\}$ and $C^\mathcal{T} \in \text{elim}_\mathcal{D}(\mathcal{P})$ of the form:

$$C^\mathcal{T} : p'(\bar{t}_n, W') \leftarrow q\text{Val}(W'), \left(\begin{array}{l} q\text{Val}(w'_j), \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner, \\ q\text{Bound}(W', \iota(\alpha), w'_j), B'_j \end{array} \right)_{j=1\dots m}.$$

Obviously, $\rho \in \text{Sol}_\mathcal{C}(\Omega)$ and $\text{dom}(\rho) = \text{var}(\Omega)$. To finish the proof we must prove $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} A'\rho \Leftarrow \Pi$. We claim that this can be done with a $\text{CHL}(\mathcal{C})$ proof tree T' whose root inference is a **DA** step of the form:

$$\frac{\begin{array}{l} (t'_i\rho == t_i\theta') \Leftarrow \Pi \quad i=1\dots n \\ (W\rho == W'\theta') \Leftarrow \Pi \\ q\text{Val}(W')\theta' \Leftarrow \Pi \\ \left(\begin{array}{l} q\text{Val}(w'_j)\theta' \Leftarrow \Pi \\ \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner\theta' \Leftarrow \Pi \\ q\text{Bound}(W', \iota(\alpha), w'_j)\theta' \Leftarrow \Pi \\ B'_j\theta' \Leftarrow \Pi \end{array} \right)_{j=1\dots m} \end{array}}{p'(\bar{t}'_n, W)\rho \Leftarrow \Pi} \quad (\spadesuit)$$

using $C^\mathcal{T}$ instantiated by the substitution $\theta' = \theta \uplus \rho_1 \uplus \dots \uplus \rho_m \uplus \{W' \mapsto \iota(d)\}$. We check that the premises of (\spadesuit) can be derived from $\text{elim}_\mathcal{D}(\mathcal{P})$ in $\text{CHL}(\mathcal{C})$:

- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} (t'_i\rho == t_i\theta') \Leftarrow \Pi$ for $i = 1 \dots n$. By construction of ρ and θ' , these are equivalent to prove $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} (t'_i == t_i\theta) \Leftarrow \Pi$ for $i = 1 \dots n$ and these hold with $\text{CHL}(\mathcal{C})$ proof trees of only one **EA** node because of $t'_i \approx_\Pi t_i\theta$, which is a consequence of the first n premises of (\clubsuit) .
- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} (W\rho == W'\theta') \Leftarrow \Pi$. By construction of ρ and θ' , this is equivalent to prove $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} (\iota(d) == \iota(d)) \Leftarrow \Pi$ which results trivial.
- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} q\text{Val}(W')\theta' \Leftarrow \Pi$. By construction of θ' , this is equivalent to prove $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} q\text{Val}(\iota(d)) \Leftarrow \Pi$. We trivially have that $\iota(d) \in \text{ran}(\iota)$. Then, by Lemma 4.2, this premise holds.
- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} q\text{Val}(w'_j)\theta' \Leftarrow \Pi$ for $j = 1 \dots m$. By construction of θ' and Lemma 4.2 we must prove, for any fixed j , that $q\text{Val}(w'_j\rho_j)$ is true in \mathcal{C} . As $\rho_j \in \text{Sol}_\mathcal{C}(\Omega_j)$ we know $\rho_j \in \text{Sol}_\mathcal{C}(q\text{Val}(w'_j))$, therefore $q\text{Val}(w'_j\rho_j)$ is trivially true in \mathcal{C} .
- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner\theta' \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . If $w_j = ?$ this results trivial. Otherwise, it amounts to $q\text{Bound}(\iota(w_j), \iota(\mathbf{t}), w'_j\rho_j)$ being true in \mathcal{C} , by construction of θ' and Lemma 4.2. As seen before, $q\text{Val}(w'_j\rho_j)$ is true in \mathcal{C} , therefore $w'_j\rho_j = \iota(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (\clubsuit) we have $w_j \leq e_j$. On the other hand, $\rho_j \in \text{Sol}_\mathcal{C}(\Omega_j)$ and, in particular, $\rho_j \in \text{Sol}_\mathcal{C}(q\text{Bound}(\iota(e_j), \iota(\mathbf{t}), w'_j))$. This, together with $w'_j\rho_j = \iota(e'_j)$, means $e_j \leq e'_j$, which with $w_j \leq e_j$ implies $w_j \leq e'_j$, i.e. $q\text{Bound}(\iota(w_j), \iota(\mathbf{t}), w'_j\rho_j)$ is true in \mathcal{C} .
- $\text{elim}_\mathcal{D}(\mathcal{P}) \vdash_\mathcal{C} q\text{Bound}(W', \iota(\alpha), w'_j)\theta' \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . By construction of θ' and Lemma 4.2, we must prove that $q\text{Bound}(\iota(d), \iota(\alpha), w'_j\rho_j)$ is true in \mathcal{C} . As seen before, $q\text{Val}(w'_j\rho_j)$ is true in \mathcal{C} , therefore $w'_j\rho_j = \iota(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (\clubsuit) we have $d \leq \alpha \circ e_j$. On the other hand, $\rho_j \in \text{Sol}_\mathcal{C}(\Omega_j)$ and, in particular, $\rho_j \in \text{Sol}_\mathcal{C}(q\text{Bound}(\iota(e_j), \iota(\mathbf{t}), w'_j))$. This, together with $w'_j\rho_j = \iota(e'_j)$, means $e_j \leq e'_j$. Now, $d \leq \alpha \circ e_j$ and $e_j \leq e'_j$ implies $d \leq \alpha \circ e'_j$, i.e. $q\text{Bound}(\iota(d), \iota(\alpha), w'_j\rho_j)$ is true in \mathcal{C} .

- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta' \Leftarrow \Pi$ for $j = 1 \dots m$. In this case, it is easy to see that $B'_j \theta' = B'_j \theta \rho_j$ by construction of θ' and because of the program transformation rules. On the other hand, proof trees T'_j proving $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta \rho_j \Leftarrow \Pi$ can be obtained by inductive hypothesis as seen before.

[2. \Rightarrow 1.] (*the transformation is sound*). We assume that T' is a $\text{CHL}(\mathcal{C})$ proof tree witnessing $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A' \rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$. We want to show the existence of a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We reason by complete induction on $\|T'\|$. There are three possible cases according to the syntactic form of the atom A' . In each case we argue how to build the desired proof tree T .

— A' is a primitive atom κ . In this case due to **TQCA** and **TPA** we can assume $A = \kappa$ and $\Omega = \{\text{qVal}(\iota(\mathbf{t})), \text{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Note that $\text{dom}(\rho) = \text{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa \varepsilon \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **PA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ with a proof tree T containing only one **QPA** node.

— A' is an equation $t == s$. In this case due to **TQCA** and **TEA** we can assume $A = (t == s)$ and $\Omega = \{\text{qVal}(\iota(\mathbf{t})), \text{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Note that $\text{dom}(\rho) = \text{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s) \varepsilon \Leftarrow \Pi$ follows $t \approx_{\Pi} s$ due to the **EA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t == s) \# d \Leftarrow \Pi$ with a proof tree T containing only one **QEA** node.

— A' is a defined atom $p'(\bar{t}'_n, W)$ with $p' \in DP^{n+1}$. In this case due to **TQCA** and **TDA** we can assume $A = p(\bar{t}'_n)$ and $\Omega = \{\text{qVal}(W), \text{qBound}(\iota(d), \iota(\mathbf{t}), W)\}$. On the other hand, T' must be rooted by a **DA** step (\spadesuit) using a clause $C^{\mathcal{T}} \in \text{elim}_{\mathcal{D}}(\mathcal{P})$ instantiated by a substitution θ' . We can assume that (\spadesuit), $C^{\mathcal{T}}$ and the corresponding clause $C \in \mathcal{P}$ have the form already displayed in [1. \Rightarrow 2.].

By construction of $C^{\mathcal{T}}$, we can assume $B_j^{\mathcal{T}} = (B'_j, w'_j)$. Let $\theta = \theta' \upharpoonright \text{var}(C)$ and $\rho_j = \theta' \upharpoonright \text{var}(w'_j)$ ($1 \geq j \geq m$). Then, due to the premises $\text{qVal}(w'_j) \theta' \Leftarrow \Pi$ of (\spadesuit) and Lemma 4.2 we can assume $e'_j \in D \setminus \{\mathbf{b}\}$ ($1 \leq j \leq m$) such that $w'_j \rho_j = \iota(e'_j)$.

To finish the proof, we must prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We claim that this can be done with a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T whose root inference is a **QDA** step of the form of (\clubsuit), as displayed in [1. \Rightarrow 2.], using clause C instantiated by θ . In the premises of this inference we choose $d_i = \mathbf{t}$ ($1 \leq i \leq n$) and $e_j = e'_j$ ($1 \leq j \leq m$). Next we check that these premises can be derived from \mathcal{P} in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ and that the side conditions are fulfilled:

- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t'_i == t_i \theta) \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. This amounts to $t'_i \approx_{\Pi} t_i \theta$ which follows from the first n premises of (\spadesuit) given that $t'_i \rho = t'_i$ and $t_i \theta' = t_i \theta$.
- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \theta \# e_j \Leftarrow \Pi$ for $j = 1 \dots m$. From $B_j^{\mathcal{T}} = (B'_j, w'_j)$ and due to rule **TQCA**, we have $((B_j \theta) \# e_j \Leftarrow \Pi)^{\mathcal{T}} = (B_j \theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\text{qVal}(w'_j), \text{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j)\}$. From the premises of (\spadesuit) and the fact that $B'_j \theta' = B'_j \theta \rho_j$ we know that $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta \rho_j \Leftarrow \Pi$ with a $\text{CHL}(\mathcal{C})$ proof tree T'_j such that $\|T'_j\| < \|T'\|$. Therefore $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \theta \# e_j \Leftarrow \Pi$ follows by inductive hypothesis provided that $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$. In fact, due to the form of Ω_j , $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $w'_j \rho_j = \iota(e'_j)$ for some e'_j such that $e_j \leq e'_j$, which is the case because of the choice of e_j .
- $e_j \geq^? w_j$ for $j = 1 \dots m$. Trivial in the case that $w_j = ?$. Otherwise they are

equivalent to $w_j \trianglelefteq e'_j$ which follow from premises $\lceil w'_j \triangleright^? \iota(w_j) \rceil \theta' \Leftarrow \Pi$ (i.e. $\lceil w'_j \rho_j \triangleright^? \iota(w_j) \rceil \Leftarrow \Pi$) of \spadesuit and Lemma 4.2.

- $d \trianglelefteq d_i$ for $i = 1 \dots n$. Trivially hold due to the choice of $d_i = \mathbf{t}$.
- $d \trianglelefteq \alpha \circ e_j$ for $j = 1 \dots m$. Note that $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ implies the existence of $d' \in D \setminus \{\mathbf{b}\}$ such that $\iota(d') = W\rho$ and $d \trianglelefteq d'$. On the other hand, $e_j = e'_j$ by choice. It suffices to prove $d' \trianglelefteq \alpha \circ e'_j$ for $j = 1 \dots m$. Premises of \spadesuit and Lemma 4.2 imply that $\mathbf{qBound}(W'\theta', \iota(\alpha), w'_j\theta')$ is true in \mathcal{C} . Moreover, $W'\theta' = W\rho = \iota(d')$ because of another premise of \spadesuit and $w'_j\theta' = \iota(e'_j)$ as explained above. Therefore $\mathbf{qBound}(W'\theta', \iota(\alpha), w'_j\theta')$ amounts to $\mathbf{qBound}(\iota(d'), \iota(\alpha), \iota(e'_j))$ which guarantees $d' \trianglelefteq \alpha \circ e'_j$ ($1 \leq j \leq m$). \square

The goal transformation correctness is established by the next theorem, which relies on the previous result.

Theorem 4.4

Let G be a goal for a QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P} such that $qVal$ and $qBound$ do not occur in G . Let $\mathcal{P}' = \text{elim}_{\mathcal{D}}(\mathcal{P})$ and $G' = \text{elim}_{\mathcal{D}}(G)$. Assume a \mathcal{C} -substitution σ , a mapping $\mu : \text{var}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and a satisfiable finite set of \mathcal{C} -constraints Π . Then, the following two statements are equivalent:

1. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.
2. $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ for some θ that verifies the following requirements:
 - (a) $\theta =_{\text{var}(G)} \sigma$,
 - (b) $\theta =_{\text{var}(G)} \mu$ and
 - (c) $W\theta \in \text{ran}(\iota)$ for each $W \in \text{var}(G') \setminus (\text{var}(G) \cup \text{var}(G))$.

Proof

As explained in Subsection 3.1 the syntax of goals in QCLP(\mathcal{D}, \mathcal{C})-programs is the same as that of goals for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs, which is described in Section 2. Therefore G , and G' due to rule **TG**, must have the following form:

$$\begin{aligned} G &: (B_j \# W_j, W_j \triangleright^? \beta_j)_{j=1 \dots m} \\ G' &: (qVal(W_j), \lceil W_j \triangleright^? \iota(\beta_j) \rceil, qVal(w'_j), qBound(W_j, \iota(\mathbf{t}), w'_j), B'_j)_{j=1 \dots m} \end{aligned}$$

with $B_j^T = (B'_j, w'_j)$ ($1 \leq j \leq m$). Note that, because of rule **TQCA**, we have $(B_j \sigma \# W_j \mu \Leftarrow \Pi)^T = (B'_j \sigma \Leftarrow \Pi, \Omega_j)$ with $\Omega_j = \{qVal(w'_j), qBound(\iota(W_j \mu), \iota(\mathbf{t}), w'_j)\}$ for $j = 1 \dots m$. We now prove each implication.

[1. \Rightarrow 2.] Let $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$. This means, by Definition 3.1, $W_j \mu \triangleright^? \beta_j$ and $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \sigma \# W_j \mu \Leftarrow \Pi$ for $j = 1 \dots m$. In these conditions, Theorem 4.3 guarantees $\mathcal{P}' \vdash_{\mathcal{C}} B'_j \sigma \rho_j \Leftarrow \Pi$ ($1 \leq j \leq m$) for some $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ such that $\text{dom}(\rho_j) = \text{var}(\Omega_j)$. It is easy to see that $\text{var}(G') \setminus (\text{var}(G) \cup \text{var}(G)) = \text{var}(\Omega_1) \uplus \dots \uplus \text{var}(\Omega_m)$. Therefore it is possible to define a substitution θ verifying $\theta =_{\text{var}(G)} \sigma$, $\theta =_{\text{var}(G)} \mu$ and $\theta =_{\text{dom}(\rho_j)} \rho_j$ ($1 \leq j \leq m$). Trivially, θ satisfies conditions 2.(a) and 2.(b). It also satisfies condition 2.(c) because for any j and any variable X such that $X \in \text{var}(\Omega_j)$, we have a constraint $qVal(X) \in \Omega_j$ implying, due to Lemma 4.2, $X\rho_j \in \text{ran}(\iota)$ (because $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$).

In order to prove $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ in the sense of Definition 3.3 we check the following items:

- By construction, θ is a \mathcal{C} -substitution.
- By the theorem's assumptions, Π is a satisfiable and finite set of \mathcal{C} -constraints.
- $\mathcal{P}' \vdash_{\mathcal{C}} A\theta \Leftarrow \Pi$ for every atom A in G' . Because of the form of G' we have to prove the following for any fixed j :
 - $\mathcal{P}' \vdash_{\mathcal{C}} qVal(W_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qVal(\iota(W_j\mu))$ being true in \mathcal{C} , which is trivial consequence of $W_j\mu \in D \setminus \{\mathbf{b}\}$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \triangleright^? \iota(\beta_j) \urcorner \theta \Leftarrow \Pi$. If $\beta_j = ?$ this becomes trivial. Otherwise, $W_j\theta = \iota(W_j\mu)$ by construction of θ , and by Lemma 4.2 it suffices to prove $qBound(\iota(\beta_j), \iota(\mathbf{t}), \iota(W_j\mu))$ is true in \mathcal{C} . This follows from $W_j\mu \triangleright^? \beta_j$, that is ensured by $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} qVal(w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qVal(w'_j\rho_j)$ being true in \mathcal{C} , that is guaranteed by $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} qBound(W_j, \iota(\mathbf{t}), w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qBound(\iota(W_j\mu), \iota(\mathbf{t}), w'_j\rho_j)$ being true in \mathcal{C} , that is also guaranteed by $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\theta \Leftarrow \Pi$. Note that, by construction of θ , $B'_j\theta = B'_j\sigma\rho_j$. On the other hand, ρ_j has been chosen above to verify $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\sigma\rho_j \Leftarrow \Pi$.

[2. \Rightarrow 1.] Let $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and assume that θ verifies 2.(a), 2.(b) and 2.(c). In order to prove $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ in the sense of Definition 3.1 we must prove the following items:

- By the theorem's assumptions, σ is a \mathcal{C} -substitution, $\mu : \text{var}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and Π is a satisfiable finite set of \mathcal{C} -constraints.
- $W_j\mu \triangleright^? \beta_j$. We reason for any fixed j . If $\beta_j = ?$ this results trivial. Otherwise, we have $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \triangleright^? \iota(\beta_j) \urcorner \theta \Leftarrow \Pi$ which, by condition 2.(b) and Lemma 4.2 amounts to $qBound(\iota(\beta_j), \iota(\mathbf{t}), \iota(W_j\mu))$ is true \mathcal{C} , i.e. $W_j\mu \triangleright \beta_j$.
- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j\sigma\#W_j\mu \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . Let ρ_j be the restriction of θ to $\text{var}(\Omega_j)$. Then, $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\sigma\rho_j \Leftarrow \Pi$ follows from $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and $B'_j\theta = B'_j\sigma\rho_j$. Therefore, $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j\sigma\#W_j\mu \Leftarrow \Pi$ follows from Theorem 5.3 provided that $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$. By Lemma 4.2 and the form of Ω_j , $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $\mathcal{P}' \vdash_{\mathcal{C}} qVal(w'_j\rho_j) \Leftarrow \Pi$ and $\mathcal{P}' \vdash_{\mathcal{C}} qBound(\iota(W_j\mu), \iota(\mathbf{t}), w'_j\rho_j) \Leftarrow \Pi$, which is true because $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and construction of ρ_j . \square

4.3 Solving SQCLP Goals

In this subsection we show that the transformations from the two previous subsections can be used to specify abstract goal solving systems for SQCLP and arguing about their correctness. In the sequel we consider a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} and a goal G for \mathcal{P} whose atoms are all relevant for \mathcal{P} . We also consider $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$, $G' = \text{elim}_{\mathcal{S}}(G)$, $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$ and $G'' = \text{elim}_{\mathcal{D}}(G')$. Due to the definition of both $\text{elim}_{\mathcal{S}}$ and $\text{elim}_{\mathcal{D}}$, we can assume:

$$\begin{aligned}
 G &: (A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m} \\
 G' &: (A_i^{\sharp} \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m} \\
 G'' &: (qVal(W_i), \ulcorner W_i \triangleright^? \iota(\beta_i) \urcorner, qVal(w'_i), qBound(W_i, \iota(\mathbf{t}), w'_i), A'_i)_{i=1 \dots m} \\
 &\quad \text{where } A_i^T = (A'_i, w'_i).
 \end{aligned}$$

In the particular case that the G is a unification problem, all atoms A_i , $i = 1 \dots m$, are equations $t_i == s_i$ and G'' is such that w'_i is a fresh CLP variable W'_i and A'_i has the form $\sim' (t_i, s_i, W'_i)$, for all $i = 1 \dots m$. Unification problems will be important for some examples when discussing our practical implementation in Section 5.

Next, we present an auxiliary result.

Lemma 4.3

Assume \mathcal{P} , G , \mathcal{P}' , G' , \mathcal{P}'' and G'' as above. Let $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$, $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ and $\theta = \sigma'\nu$. Then $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$. Moreover, $W\theta \in \text{ran}(\imath)$ for every $W \in \text{var}(G'') \setminus \text{var}(G)$.²

Proof

Consider an arbitrary atom A'' occurring in G'' . Because of $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P} \vdash_{\mathcal{C}} A''\sigma' \Leftarrow \Pi$. On the other hand, because of $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ we have $\emptyset \models_{\mathcal{C}} \Pi\nu$ and therefore also $\Pi \models_{\mathcal{C}} \Pi\nu$. This and Definition 3.1(4) of (Rodríguez-Artalejo and Romero-Díaz 2010b) ensure $A''\sigma' \Leftarrow \Pi \succ_{\mathcal{C}} A''\sigma'\nu \Leftarrow \Pi$, i.e. $A''\sigma' \Leftarrow \Pi \succ_{\mathcal{C}} A''\theta \Leftarrow \Pi$. This fact, $\mathcal{P}'' \vdash_{\mathcal{C}} A''\sigma' \Leftarrow \Pi$ and the Entailment Property for Programs in $\text{CLP}(\mathcal{C})$ imply $\mathcal{P}'' \vdash_{\mathcal{C}} A''\theta \Leftarrow \Pi$. Therefore, $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$.

Consider now any $W \in \text{var}(G'') \setminus \text{var}(G)$. By construction of G'' , one of the atoms occurring in G'' is $q\text{Val}(W)$. Then, due to $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P}'' \vdash_{\mathcal{C}} q\text{Val}(W\sigma') \Leftarrow \Pi$. Because of Lemma 3.1(1) this implies $\Pi \models_{\mathcal{C}} q\text{Val}(W\sigma')$, i.e. $\text{Sol}_{\mathcal{C}}(\Pi) \subseteq \text{Sol}_{\mathcal{C}}(q\text{Val}(W\sigma'))$. Since $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ we get $\nu \in \text{Sol}_{\mathcal{C}}(q\text{Val}(W\sigma'))$, i.e. $W\sigma'\nu \in \text{ran}(\imath)$. Since $W\sigma'\nu = W\theta$, we are done. \square

Now, we can explain how to define an abstract goal solving system for SQCLP from a given abstract goal solving system for CLP.

Definition 4.4

Let \mathcal{CA}'' be an abstract goal solving system for $\text{CLP}(\mathcal{C})$ (in the sense of Definition 3.4). Then we define \mathcal{CA} as an abstract goal solving system for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ (in the sense of Definition 2.3) that works as follows:

1. Given a goal G for the $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} , consider \mathcal{P}' , G' , \mathcal{P}'' and G'' as explained at the beginning of the subsection.
2. For each $\langle \sigma', \Pi \rangle \in \mathcal{CA}''_{\mathcal{P}''}(G'')$ and for any $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$, let $\langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$, where $\theta = \sigma'\nu$, $\sigma = \theta \upharpoonright \text{var}(G)$ and $\mu = \theta \iota^{-1} \upharpoonright \text{var}(G)$. Note that μ is well-defined thanks to Lemma 4.3.
3. All the computed answers belonging to $\mathcal{CA}_{\mathcal{P}}(G)$ are obtained as described in the previous item. \square

The next theorem ensures that \mathcal{CA} is correct provided that \mathcal{CA}'' is also correct. The proof relies on the semantic results of the two previous subsections.

Theorem 4.5 (Correct Abstract Goal Solving Systems for SQCLP)

Let \mathcal{CA} be obtained from \mathcal{CA}'' as in the previous definition. Assume that \mathcal{CA}'' is correct as specified in Definition 3.4(3). Then \mathcal{CA} is correct as specified in Definition 2.3(4).

² Note that $\text{var}(G) \subseteq \text{var}(G'') \setminus \text{var}(G)$.

Proof

We separately prove that \mathcal{CA} is *sound* and *weakly complete*.

— \mathcal{CA} is *sound*. Assume $\langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$. We must prove that $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$. Because of Definition 4.4 there exist $\langle \sigma', \Pi \rangle \in \mathcal{CA}''_{\mathcal{P}''}(G'')$ and $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$ such that $\sigma = \theta \upharpoonright \text{var}(G)$ and $\mu = \theta \iota^{-1} \upharpoonright \text{var}(G)$ with $\theta = \sigma' \nu$. By the soundness of \mathcal{CA}'' we get $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$. Moreover, because of Lemma 4.3 we have $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ and $W\theta \in \text{ran}(\iota)$ for every $W \in \text{var}(G'') \setminus \text{var}(G)$. Note that:

- $\theta =_{\text{var}(G')} \sigma$. This follows from $\text{var}(G') = \text{var}(G)$ and the construction of σ .
- $\theta =_{\text{var}(G')} \mu$. This follows from $\text{var}(G') = \text{var}(G)$ and $\theta =_{\text{var}(G)} \mu$, that is obvious from the construction of μ .
- $W\theta \in \text{ran}(\iota)$ for each $W \in \text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G'))$. This is a consequence of Lemma 4.3 since $\text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G')) \subseteq \text{var}(G'') \setminus \text{var}(G')$ and $\text{var}(G') = \text{var}(G)$.

From the previous items and Theorem 4.4 we get $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$, which trivially implies $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ because of Theorem 4.2.

— \mathcal{CA} is *weakly complete*. Let $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ be a ground solution for G w.r.t. \mathcal{P} . We must prove that it is subsumed by some computed answer $\langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$. By Theorem 4.2 we have that $\langle \eta, \rho, \emptyset \rangle$ is also a ground solution for G' w.r.t. \mathcal{P}' . Then by Theorem 4.4 we get $\langle \eta', \emptyset \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ for some η' such that

- (1) $\eta' =_{\text{var}(G')} \eta$,
- (2) $\eta' =_{\text{var}(G')} \rho$ and hence $\eta'(\iota^{-1}) =_{\text{var}(G')} \rho$, and
- $W\eta' \in \text{ran}(\iota)$ for each $W \in \text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G'))$ (i.e. $w'_i \eta' \in \text{ran}(\iota)$ for each $i = 1 \dots m$ such that w'_i is a variable).

By construction of η' , it is clear that $\langle \eta', \emptyset \rangle$ is ground. Now, by the weak completeness of \mathcal{CA}'' , there is some computed answer $\langle \sigma', \Pi \rangle \in \mathcal{CA}''_{\mathcal{P}''}(G'')$ subsuming $\langle \eta', \emptyset \rangle$ in the sense of Definition 3.3(5), therefore satisfying:

- (3) there is some $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$, such that
- (4) $\eta' =_{\text{var}(G'')} \sigma' \nu$.

Because of Definition 4.4 one can build a computed answer $\langle \sigma, \mu, \Pi \rangle \in \mathcal{CA}_{\mathcal{P}}(G)$ as follows:

- (5) $\sigma = \sigma' \nu \upharpoonright \text{var}(G)$
- (6) $\mu = \sigma' \nu \iota^{-1} \upharpoonright \text{var}(G)$

We now check that $\langle \sigma, \mu, \Pi \rangle$ subsumes $\langle \eta, \rho, \emptyset \rangle$ in the sense of Definition 2.2(4):

- $W_i \rho \leq W_i \mu$ and even $W_i \rho = W_i \mu$ because:

$$W_i \rho =_{(2)} W_i \eta'(\iota^{-1}) =_{(4)} W_i \sigma' \nu(\iota^{-1}) =_{(6)} W_i \mu .$$

- $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$ by (3) and, moreover, for any $X \in \text{var}(G)$:

$$X\eta =_{(1)} X\eta' =_{(4)} X\sigma' \nu =_{(\dagger)} X\sigma' \nu \nu =_{(5)} X\sigma \nu$$

therefore $\eta =_{\text{var}(G)} \sigma \nu$.

The step (\dagger) is justified because $\nu \in \text{Val}_{\mathcal{L}}$ implies $\nu = \nu \nu$. \square

As an immediate consequence of Theorem 4.5 and Lemma 2.1, we obtain:

Corollary 4.1 (Flexibly Correct Abstract Goal Solving Systems for SQCLP)

Let \mathcal{CA} be obtained from \mathcal{CA}'' as in the Definition 4.4. Assume that \mathcal{CA}'' is correct as specified in Definition 3.4(3). Then any flexible restriction \mathcal{FCA} of \mathcal{CA} is correct in the flexible sense as specified in Definition 2.3(5). \square

5 A Practical Implementation

This section is devoted to the more practical aspects of the SQCLP programming scheme. We present a **Prolog**-based prototype system that relies on the transformation techniques from Section 4 and supports several useful SQCLP instances. The presentation is developed in three subsections. Subsection 5.1 discusses in some detail how to bridge the gap between the abstract goal solving systems for SQCLP discussed in Subsection 4.3 and a practical **Prolog**-based implementation. Subsection 5.2 gives a user-oriented presentation of our prototype implementation, explaining how to write programs and how to solve goals. Finally, in Subsection 5.3 we study the unavoidable overload caused by the implementation of qualification and proximity relations in our system. The overload is shown in experimental results on the execution of some SQCLP programs which make only a trivial use of qualification and proximity.

5.1 SQCLP over a CLP Prolog System

Our aim is to implement a goal solving system for SQCLP on top of an available CLP **Prolog** system, taking the definitions and results from Subsection 4.3 as a theoretical guideline. Therefore, given a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} and a goal G for \mathcal{P} , the following steps should be carried out:

- (i) Apply the transformation $\text{elim}_{\mathcal{S}}$ specified in Definition 4.2, obtaining the QCLP(\mathcal{D}, \mathcal{C}) program $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ and the QCLP(\mathcal{D}, \mathcal{C}) goal $G' = \text{elim}_{\mathcal{S}}(G)$, where G and G' are as displayed at the beginning of Section 4.3, \mathcal{P}' is of the form $EQ_{\mathcal{S}} \cup \hat{\mathcal{P}}_{\mathcal{S}}$, $EQ_{\mathcal{S}}$ is obtained following Definition 4.1 and $\hat{\mathcal{P}}_{\mathcal{S}}$ is obtained following Definition 4.2(3,2).
- (ii) Apply the transformation $\text{elim}_{\mathcal{D}}$ specified in Definition 4.3, obtaining the CLP(\mathcal{C})-program $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$ and the CLP(\mathcal{C})-goal $G'' = \text{elim}_{\mathcal{D}}(G')$, where G' and G'' (obtained from G' by the goal transformation rules shown in Figure 5) are as displayed at the beginning of Section 4.3 and \mathcal{P}'' is built according to Definition 4.3, by adding the two clauses of the program $E_{\mathcal{D}}$ to the result of applying the program transformation rules shown in Figure 5 to the program \mathcal{P}' . In particular, \mathcal{P}'' includes as a subset the set $EQ'_{\mathcal{S}}$ of CLP(\mathcal{C})-clauses obtained by applying the transformation rules from Figure 5 to the set of QCLP(\mathcal{D}, \mathcal{C})-clauses $EQ_{\mathcal{S}}$.
- (iii) Use the available CLP **Prolog** system to compute answers for the CLP goal G'' by executing the CLP program \mathcal{P}'' .

Following these steps literally would lead to a set of computed answers representing the behaviour of the abstract goal solving system \mathcal{CA} from Definition 4.4³, whose correctness has been proved in Theorem 4.5. Therefore, the resulting implementation would be correct—i.e. both sound and weakly complete—in the sense of Definition 2.3, except for the unavoidable failures in completeness due to **Prolog**'s computation strategy and the incompleteness of the constraint solvers provided by practical CLP **Prolog** systems.

However, our **Prolog**-based implementation—presented in Subsection 5.2—differs from the literal application of step (ii) in some aspects concerning an optimized implementation of the CLP clauses in the sets $E_{\mathcal{D}}$ and $EQ'_{\mathcal{S}}$. In the rest of this subsection we explain the optimizations and we discuss their influence on the correctness (i.e. soundness and weak completeness) of goal solving. Subsections 5.1.1 and 5.1.2 below present some straightforward optimizations of the CLP clauses in $E_{\mathcal{D}}$ and $EQ'_{\mathcal{S}}$, respectively, while Subsection 5.1.3 discusses three possible **Prolog** implementations of the optimized set $EQ'_{\mathcal{S}}$ obtained in Subsection 5.1.2: a naïve one—called **(A)**—that causes very inefficient computations and is not supported by our system; and two optimized ones—called **(B)** and **(C)**—with a better computational behaviour, which are supported by our system.

5.1.1 Optimization of the $E_{\mathcal{D}}$ clauses

Here we present a straightforward optimization of $E_{\mathcal{D}}$ that does not modify the set of computed answers, thus preserving correctness of goal solving. As explained at the beginning of Section 4.2, the set $E_{\mathcal{D}}$ contains CLP clauses for two predicates $qVal$ (unary) and $qBound$ (ternary) which allow to represent qualification values from \mathcal{D} and the behaviour of \mathcal{D} 's attenuation operator \circ by means of \mathcal{C} -constraints. Recall Example 4.3, showing the clauses in $E_{\mathcal{D}}$ for three significative choices of \mathcal{D} , namely \mathcal{U} , \mathcal{W} and $\mathcal{U} \otimes \mathcal{W}$.

Our prototype system for SQCLP programming supports SQCLP instances of the form $SQCLP(\mathcal{S}, \mathcal{D}, \mathcal{R})$, where \mathcal{R} is the real constraint domain and \mathcal{D} is any qualification domain that can be built from \mathcal{B} , \mathcal{U} and \mathcal{W} by means of the strict cartesian product operation \otimes . Instead of using a different set $E_{\mathcal{D}}$ for each choice of \mathcal{D} supported by the system, our implementation uses a single set of **Prolog** clauses for two predicates **qVal** (binary) and **qBound** (quaternary), whose additional argument w.r.t. $qVal$ and $qBound$ is used to encode a representation of \mathcal{D} in the following way: \mathcal{B} , \mathcal{U} and \mathcal{W} are encoded as **b**, **u** and **w**, respectively; while $\mathcal{D}_1 \otimes \mathcal{D}_2$ is encoded as an ordered pair built from the encodings of \mathcal{D}_1 and \mathcal{D}_2 . The set of **Prolog** clauses for **qVal** and **qBound** used in our implementation is as follows⁴:

$$\begin{aligned} E_1 \quad & \text{qVal}(\mathbf{b}, 1). \\ E_2 \quad & \text{qVal}(\mathbf{u}, \mathbf{X}) \text{ :- } \{ \mathbf{X} > 0, \mathbf{X} \leq 1 \}. \end{aligned}$$

³ Each answer $\langle \sigma', \Pi \rangle$ produced by the CLP system and shown to the user in step (iii) serves as a compact representation of all answers of the form $\langle \sigma, \mu, \Pi \rangle \in \mathcal{CAP}(G)$, where $\theta = \sigma' \nu$, $\sigma = \theta|_{\text{var}(G)}$, $\mu = \theta \iota^{-1}|_{\text{var}(G)}$, and $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ ranges over the solutions of Π .

⁴ The semantic correctness of these clauses is obvious from the definition of \mathcal{B} , \mathcal{U} , \mathcal{W} and \otimes ; see (Rodríguez-Artalejo and Romero-Díaz 2010b) for details.

$$\begin{aligned}
E_3 \quad & \text{qVal}(\mathbf{w}, X) :- \{ X > 0 \}. \\
E_4 \quad & \text{qVal}((D_1, D_2), (X_1, X_2)) :- \text{qVal}(D_1, X_1), \text{qVal}(D_2, X_2). \\
E_5 \quad & \text{qBound}(\mathbf{b}, 1, 1, 1). \\
E_6 \quad & \text{qBound}(u, X, Y, Z) :- \{ X = Y * Z \}. \\
E_7 \quad & \text{qBound}(\mathbf{w}, X, Y, Z) :- \{ X > Y + Z \}. \\
E_8 \quad & \text{qBound}((D_1, D_2), (X_1, X_2), (Y_1, Y_2), (Z_1, Z_2)) :- \text{qBound}(D_1, X_1, Y_1, Z_1), \\
& \quad \text{qBound}(D_2, X_2, Y_2, Z_2).
\end{aligned}$$

Therefore, calls such as $qVal(X)$ and $qBound(X, Y, Z)$ to the $E_{\mathcal{D}}$ predicates are implemented as $qVal(b, X)$ and $qBound(b, X, Y, Z)$, if $\mathcal{D} = \mathcal{B}$; as $qVal(u, X)$ and $qBound(u, X, Y, Z)$, if $\mathcal{D} = \mathcal{U}$; as $qVal(w, X)$ and $qBound(w, X, Y, Z)$, if $\mathcal{D} = \mathcal{W}$; as $qVal((u, w), X)$ and $qBound((u, w), X, Y, Z)$, if $\mathcal{D} = \mathcal{U} \otimes \mathcal{W}$; etc.

In order to simplify the presentation, in the rest of Subsection 5.1 we will omit the optimization just discussed, considering $E_{\mathcal{D}}$ as a set of CLP clauses for a unary predicate $qVal$ and a ternary predicate $qBound$ corresponding to some fixed choice of \mathcal{D} .

5.1.2 Optimization of the EQ'_S clauses

Now we present a simple optimization of the CLP clauses in EQ'_S . Recall that EQ'_S is the set of CLP(\mathcal{C})-clauses obtained by applying the transformation rules in Figure 5 to the set EQ_S of QCLP(\mathcal{D}, \mathcal{C})-clauses built according to Definition 4.1. Therefore, EQ'_S consists of CLP clauses of the following forms:

$$\begin{aligned}
EQ_1 \quad & \sim'(X, Y, W) \leftarrow qVal(W), X = Y \\
EQ_2 \quad & \sim'(u, u', W) \leftarrow qVal(W), qVal(W'), qBound(W, \mathbf{t}, W'), pay'_\lambda(W') \\
EQ_3 \quad & \sim'(c(\overline{X}_n), c'(\overline{Y}_n), W) \leftarrow qVal(W), \\
& \quad qVal(W'), qBound(W, \mathbf{t}, W'), pay'_\lambda(W'), \\
& \quad qVal(W_1), qBound(W, \mathbf{t}, W_1), \sim'(X_1, Y_1, W_1), \\
& \quad \dots \\
& \quad qVal(W_n), qBound(W, \mathbf{t}, W_n), \sim'(X_n, Y_n, W_n) \\
EQ_4 \quad & pay'_\lambda(W) \leftarrow qVal(W), qBound(W, \lambda, \mathbf{t})
\end{aligned}$$

where clauses of the form EQ_2 are one for each $u, u' \in B_{\mathcal{C}}$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$; EQ_3 are one for each $c, c' \in DC^n$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{t}$ (including the case $c = c'$, $\mathcal{S}(c, c') = \mathbf{t} \neq \mathbf{b}$); and EQ_4 are one for each pay_λ such that there exist $x, y \in S$ satisfying $\mathcal{S}(x, y) = \lambda \neq \mathbf{b}$.

By unfolding the calls to predicates pay_λ occurring in the bodies of clauses EQ_2 and EQ_3 with respect to the clauses EQ_4 defining pay_λ , all the occurrences of pay_λ —including clauses EQ_4 themselves—can be removed. Moreover, the calls to the predicates $qVal$ and $qBound$ occurring in the results of unfolding clauses EQ_2 and EQ_3 can be further simplified. Let us illustrate this process with a clause of the form EQ_2 . The original clause is:

$$\sim'(u, u', W) \leftarrow qVal(W), qVal(W'), qBound(W, \mathbf{t}, W'), pay'_\lambda(W')$$

which can be transformed into the equivalent clause:

$$\sim'(u, u', W) \leftarrow qVal(W), qVal(W'), qBound(W, \mathbf{t}, W'), qVal(W'), qBound(W', \lambda, \mathbf{t})$$

by unfolding the predicate call $pay'_\lambda(W')$ occurring in its body. Next, removing one of the two repeated predicate calls $qVal(W')$ in the new body yields the equivalent clause:

$$\sim'(u, u', W) \leftarrow qVal(W), qVal(W'), qBound(W, \mathbf{t}, W'), qBound(W', \lambda, \mathbf{t})$$

Observing the last clause we note:

- The body is logically equivalent to the following formulation:

$$qVal(W) \wedge \exists W' (qVal(W') \wedge qBound(W, \mathbf{t}, W') \wedge qBound(W', \lambda, \mathbf{t}))$$

- The second conjunct above encodes the statement

$$\exists W' (W' \in D \setminus \{\mathbf{b}\} \wedge W \leq \mathbf{t} \circ W' \wedge W' \leq \lambda \circ \mathbf{t})$$

Due to the transitivity of \leq , this is equivalent to $W \leq \lambda$ and can be encoded as $qBound(W, \mathbf{t}, \lambda)$.

Therefore, the last clause is equivalent to the following optimized form:

$$\sim'(u, u', W) \leftarrow qVal(W), qBound(W, \mathbf{t}, \lambda).$$

Performing a similar transformation for clauses EQ_3 and removing clauses EQ_4 leads to an optimized version of the set EQ'_S consisting of clauses of the following forms:

$$\begin{aligned} EQ_1 \quad & \sim'(X, Y, W) \leftarrow qVal(W), X=Y \\ EQ_2 \quad & \sim'(u, u', W) \leftarrow qVal(W), qBound(W, \mathbf{t}, \lambda) \\ EQ_3 \quad & \sim'(c(\bar{X}_n), c'(\bar{Y}_n), W) \leftarrow qVal(W), qBound(W, \mathbf{t}, \lambda), \\ & \quad qVal(W_1), qBound(W, \mathbf{t}, W_1), \sim'(X_1, Y_1, W_1), \\ & \quad \dots \\ & \quad qVal(W_n), qBound(W, \mathbf{t}, W_n), \sim'(X_n, Y_n, W_n) \end{aligned}$$

Note that a similar optimization—unfolding of calls to predicates pay_λ followed by simplification of calls to predicates $qVal$ and $qBound$ —can be done for all those clauses in \mathcal{P}'' which include calls to predicates pay_λ in their bodies. The same is true for goals. All CLP(\mathcal{C})-goals G'' occurring in subsequent examples will be displayed in the optimized form.

Clearly, the optimizations described in this subsection do not modify the set of computed answers. Therefore, correctness of goal solving is preserved.

5.1.3 Prolog Implementation of the optimized EQ'_S clauses

The optimized version of EQ'_S displayed near the end of the previous subsection just consists of clauses for the predicate \sim' . In the sequel, the notation EQ'_S will refer to this optimized version. We will consider in turn three possible Prolog implementations of the EQ'_S clauses, called **(A)**, **(B)** and **(C)**. We will give reasons for discarding implementation **(A)**—not supported by our prototype system—and we will discuss the properties of implementations **(B)** and **(C)**—both supported by

our system—concerning correctness of goal solving. At some points, our discussion will refer to Example 2.2.

The **Prolog** code displayed below is a naïve implementation of EQ'_S . Its structure does not directly resemble the clauses in the set EQ'_S , but it serves as a first step towards the more practical implementations **(B)** and **(C)** discussed below.

(A) Naïve implementation of \sim' .

```

s1   $\sim'(X, Y, W) :- \text{var}(X), \text{var}(Y), \sim'_v(X, Y, W).$ 
s2   $\sim'(X, Y, W) :- \text{var}(X), \text{nonvar}(Y), \sim'_v(X, Y, W).$ 
s3   $\sim'(X, Y, W) :- \text{nonvar}(X), \text{var}(Y), \sim'_v(X, Y, W).$ 
s4   $\sim'(X, Y, W) :- \text{nonvar}(X), \text{nonvar}(Y), \sim'_c(X, Y, W).$ 
v1   $\sim'_v(X, Y, W) :- \text{qVal}(W), X = Y.$ 
v2   $\sim'_v(X, Y, W) :- \sim'_c(X, Y, W).$ 
c1   $\sim'_c(u, u', W) :- \text{qVal}(W), \text{qBound}(W, t, \lambda).$ 
c2   $\sim'_c(c(X_1, \dots, X_n), c'(Y_1, \dots, Y_n), W) :- \text{qVal}(W), \text{qBound}(W, t, \lambda),$ 
       $\text{qVal}(W_1), \text{qBound}(W, t, W_1), \sim'(X_1, Y_1, W_1),$ 
       $\dots$ 
       $\text{qVal}(W_n), \text{qBound}(W, t, W_n), \sim'(X_n, Y_n, W_n).$ 

```

where clauses of the form C_1 are one for each $u, u' \in B_C$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$, and clauses of the form C_2 are one for each $c, c' \in DC^n$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$ (including the case $c = c', \mathcal{S}(c, c') = \mathbf{t} \neq \mathbf{b}$).

We claim that both **(A)** and EQ'_S compute the same solutions. In order to understand that, consider the behaviour of **(A)** when an atom of the form $\sim'(X, Y, W)$ is to be solved. The **Prolog** metapredicates **var** and **nonvar** are first used to distinguish four possible cases concerning X and Y . If either X or Y , or both, is a variable—more precisely, it is bound to a variable at execution time—then a first answer is computed by clause V_1 by performing the normal **Prolog** unification of X and Y , and clause V_2 can invoke clauses C_1 and C_2 in order to compute additional answers corresponding to non-syntactical unifiers of (the terms bound to) X and Y modulo the proximity relation \mathcal{S} . If neither X and Y is (bound to) a variable, then clauses C_1 and C_2 will compute answers corresponding to the unifiers of (the terms bound to) X and Y modulo \mathcal{S} . Each computed answer also includes the appropriate constraints for the variable W , thus representing a qualification level.

As far as permitted by **Prolog**'s computation strategy—which solves goal atoms from left to right and tries to apply program clauses in their textual order—the answers computed by **(A)** are the same as those which would be computed by EQ'_S . Therefore, the naïve implementation guarantees soundness and weak completeness of goal solving—recall Definition 2.3—except for failures in completeness due to **Prolog**'s computation strategy.

As an illustration, let us show the behaviour of implementation **(A)** when solving the unification problem of Example 2.2:

Example 5.1

Let $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$, \mathcal{P} and G be as in Example 2.2. Then, G'' is the following $\text{CLP}(\mathcal{R})$ -

goal:

$$\begin{aligned} & qVal(W_1), qBound(0.8, 1, W_1), \sim'(Y, X, W_1), \\ & qVal(W_2), qBound(0.8, 1, W_2), \sim'(X, b, W_2), \\ & qVal(W_3), qBound(0.8, 1, W_3), \sim'(Y, c, W_3) \end{aligned}$$

In this simple example, the **Prolog**'s computation strategy causes no loss of completeness, and the naïve **Prolog** implementation of \sim' allows to compute sol_i ($i = 1, 2, 3$) as answers for G'' . \square

However, **Prolog**'s computation strategy leads in general to a very poor computational behaviour when executing the **Prolog** code **(A)** for predicate \sim' . As justification for this claim, we argue as follows:

1. Solving a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-goal G yields to solving the translated CLP(\mathcal{C})-goal G'' . As seen in Example 5.1, G'' may include subgoals such as

$$(\star) \quad qVal(W), qBound(d, \mathbf{t}, W), \sim'(X, Y, W)$$

with $d \in D \setminus \{\mathbf{b}\}$. Solving such a subgoal in a **Prolog** system that relies on the naïve code **(A)** for the predicate \sim' may lead to compute infinitely many answers. For instance, assuming a proximity relation \mathcal{S} such that $\mathcal{S}(c, d) = \mathcal{S}(d, c) = \lambda$ with $c, d \in DC^1$, the **Prolog** code **(A)** will include, among others, the following clauses

$$\begin{aligned} s_1 \quad & \sim'(X, Y, W) :- \text{var}(X), \text{var}(Y), \sim'_v(X, Y, W). \\ v_1 \quad & \sim'_v(X, Y, W) :- qVal(W), X = Y. \\ v_2 \quad & \sim'_v(X, Y, W) :- \sim'_c(X, Y, W). \\ c_2 \quad & \sim'_c(c(X_1), c(Y_1), W) :- qVal(W), qBound(W, \mathbf{t}, \mathbf{t}), \\ & qVal(W_1), qBound(W, \mathbf{t}, W_1), \sim'(X_1, Y_1, W_1). \end{aligned}$$

whose application, in the given textual order, yields to the computation of the following answers:

- $\langle \{Y \mapsto X\}, \{W \mapsto \mathbf{t}\}, \emptyset \rangle$
- $\langle \{X \mapsto c(A), Y \mapsto c(A)\}, \{W \mapsto \mathbf{t}\}, \emptyset \rangle$
- $\langle \{X \mapsto c(c(A)), Y \mapsto c(c(A))\}, \{W \mapsto \mathbf{t}\}, \emptyset \rangle$
- ...

2. Due to the infinite sequence of **Prolog** computed answers for the goal (\star) shown in the previous item, **Prolog** never comes to computing other valid solutions for (\star) involving data constructors other than c . More concretely, due to $\mathcal{S}(c, d) = \mathcal{S}(d, c) = \lambda$, the **Prolog** code **(A)** must include clauses of the following form:

$$\begin{aligned} c_{2.1} \quad & \sim'_c(c(X_1), d(Y_1), W) :- qVal(W), qBound(W, \mathbf{t}, \lambda), [\dots]. \\ c_{2.2} \quad & \sim'_c(d(X_1), c(Y_1), W) :- qVal(W), qBound(W, \mathbf{t}, \lambda), [\dots]. \\ c_{2.3} \quad & \sim'_c(d(X_1), d(Y_1), W) :- qVal(W), qBound(W, \mathbf{t}, \mathbf{t}), [\dots]. \end{aligned}$$

If all these clauses happen to occur after the clause C_2 of item (1) in the textual order, **Prolog**'s computation strategy will never come to the point of trying to apply them to compute answers for (\star) .

Items (1) and (2) above show that the naïve implementation of \sim' is inclined to go into infinite computations which may produce infinitely many computed answers of a certain shape, while failing to compute some other answers needed for completeness. In situations a bit more complex than the one considered in items (1) and (2) above, this unfortunate behaviour can lead to failure (i.e., compute no answer at all) for goals which do have solutions, as illustrated by the following example:

Example 5.2 (Failure of the naïve implementation of \sim')

Consider the admissible triple $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$ where \mathcal{S} is a proximity relation such that: $\mathcal{S}(f, g) = \mathcal{S}(g, f) = 0.8$ and $\mathcal{S}(g, h) = \mathcal{S}(h, g) = 0.8$ where $f, g, h \in DC^1$. Assume also a constant $a \in DC^0$. Let \mathcal{P} be the empty program and let G be the following unification problem:

$$(X == f(Y))\#W_1, (X == h(Z))\#W_2 \parallel W_1 \geq 0.5, W_2 \geq 0.5$$

Then, using the naïve implementation **(A)** of \sim' leads to the following Prolog code for the CLP(\mathcal{R})-program \mathcal{P}'' :

```

1  qVal(X) :- {X > 0, X =< 1}.
2  qBound(X,Y,Z) :- {X =< Y * Z}.
3   $\sim'$ (X,Y,W) :- var(X), var(Y),  $\sim'_v$ (X,Y,Z).
4   $\sim'$ (X,Y,W) :- var(X), nonvar(Y),  $\sim'_v$ (X,Y,Z).
5   $\sim'$ (X,Y,W) :- nonvar(X), var(Y),  $\sim'_v$ (X,Y,Z).
6   $\sim'$ (X,Y,W) :- nonvar(X), nonvar(Y),  $\sim'_c$ (X,Y,Z).
7   $\sim'_v$ (X,Y,W) :- qVal(W), X = Y.
8   $\sim'_v$ (X,Y,W) :-  $\sim'_c$ (X,Y,W).
9   $\sim'_c$ (a,a,W) :- qVal(W), qBound(W,1,1).
10  $\sim'_c$ (f(X),f(Y),W) :- qVal(W), qBound(W,1,1), [...].
11  $\sim'_c$ (g(X),g(Y),W) :- qVal(W), qBound(W,1,1), [...].
12  $\sim'_c$ (h(X),h(Y),W) :- qVal(W), qBound(W,1,1), [...].
13  $\sim'_c$ (f(X),g(Y),W) :- qVal(W), qBound(W,1,0.8), [...].
14  $\sim'_c$ (g(X),f(Y),W) :- qVal(W), qBound(W,1,0.8), [...].
15  $\sim'_c$ (g(X),h(Y),W) :- qVal(W), qBound(W,1,0.8), [...].
16  $\sim'_c$ (h(X),g(Y),W) :- qVal(W), qBound(W,1,0.8), [...].

```

where the ellipsis “[...]” stands for “qVal(W₁), qBound(W,1,W₁), \sim' (X,Y,W₁)”. Note that the definitions for the program transformations do not require any specific order for the final clauses. On the other hand, G'' becomes the CLP(\mathcal{R})-goal:

$$qVal(W_1), qBound(0.5,1,W_1), \sim'(X,f(Y),W_1), \\ qVal(W_2), qBound(0.5,1,W_2), \sim'(X,h(Z),W_2)$$

When trying to solve G'' using the naïve implementation of \sim' , Prolog successively computes infinitely many answers for the subgoal consisting of the first three atoms, none of which can be continued to a successful answer of the whole goal. Therefore, the overall global computation fails. Since G has valid solutions such as

$$\langle \{X \mapsto g(Y), Z \mapsto Y\}, \{W_1 \mapsto 0.8, W_2 \mapsto 0.8\}, \emptyset \rangle$$

and also valid ground solutions such as

$$\langle \{X \mapsto g(a), Y \mapsto a, Z \mapsto a\}, \{W_1 \mapsto 0.8, W_2 \mapsto 0.8\}, \emptyset \rangle$$

the incompleteness of **Prolog**'s computation strategy causes weak completeness of SQCLP goal solving to fail in this example. \square

The problems just explained have a big impact concerning not only completeness, but also efficiency. Therefore, our **Prolog**-based system for SQCLP programming discards the naïve implementation of the EQ'_S clauses. Instead, the following **Prolog** code for predicate \sim' is used by our system:

(B) Practical implementation of \sim' intended for arbitrary proximity relations.

```

s1   $\sim'(X,Y,W) :- \text{var}(X), \text{var}(Y), \sim'_v(X,Y,W).$ 
s2   $\sim'(X,Y,W) :- \text{var}(X), \text{nonvar}(Y), \sim'_c(X,Y,W).$ 
s3   $\sim'(X,Y,W) :- \text{nonvar}(X), \text{var}(Y), \sim'_c(X,Y,W).$ 
s4   $\sim'(X,Y,W) :- \text{nonvar}(X), \text{nonvar}(Y), \sim'_c(X,Y,W).$ 

v1   $\sim'_v(X,Y,W) :- \text{qVal}(W), X = Y.$ 

c1   $\sim'_c(u,u',W) :- \text{qVal}(W), \text{qBound}(W,t,\lambda).$ 
c2   $\sim'_c(c(X_1,\dots,X_n),c'(Y_1,\dots,Y_n),W) :- \text{qVal}(W), \text{qBound}(W,t,\lambda),$ 
       $\text{qVal}(W_1), \text{qBound}(W,t,W_1), \sim'(X_1,Y_1,W_1),$ 
       $\dots$ 
       $\text{qVal}(W_n), \text{qBound}(W,t,W_n), \sim'(X_n,Y_n,W_n).$ 

```

where, again, clauses of the form C_1 are one for each $u, u' \in B_C$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$; and C_2 are one for each $c, c' \in DC^m$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$ (including the case $c = c', \mathcal{S}(c, c') = \mathbf{t} \neq \mathbf{b}$).

The difference between the implementation **(B)** and the implementation **(A)** is the use of the predicate call $\sim'_c(X,Y,W)$ instead of $\sim'_v(X,Y,W)$ at the bodies of clauses S_2 and S_3 and the removal of clause V_2 . These two changes have the effect of avoiding the enumeration of solutions when an equality between two variables is being solved. For example, for the goal (\star) shown above, the **Prolog** code **(B)** just computes the answer $\langle \{Y \mapsto X\}, \{W \mapsto \mathbf{t}\}, \emptyset \rangle$, while the **Prolog** code **(A)** infinitely enumerates many computed answers, as explained before. In general, answers computed by the implementation **(B)** of \sim' correspond to a more limited enumeration of solutions, depending on the data constructor symbols present in the goal. The following example illustrates the behaviour of implementation **(B)** in a more interesting case:

Example 5.3 (Avoiding infinite computations)

Consider the admissible triple $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$ of Example 5.2, and let \mathcal{P} be the empty program. Recall the goal G'' from Example 5.2:

$$\text{qVal}(W_1), \text{qBound}(0.5, 1, W_1), \sim'(X, f(Y), W_1), \\ \text{qVal}(W_2), \text{qBound}(0.5, 1, W_2), \sim'(X, h(Z), W_2)$$

Then, for the subgoal consisting of the first three atoms of G'' the answers computed by **Prolog** when the predicate \sim' is implemented as in **(B)** are:

$$\langle \{X \mapsto f(Y)\}, \{W_1 \mapsto 1\}, \emptyset \rangle \text{ and } \langle \{X \mapsto g(Y)\}, \{W_1 \mapsto 0.8\}, \emptyset \rangle.$$

And for the whole goal G'' , the only computed answer is:

$$\langle \{X \mapsto g(Y), Z \mapsto Y\}, \{W_1 \mapsto 0.8, W_2 \mapsto 0.8\}, \emptyset \rangle. \quad \square$$

Note, however, that the optimization achieved by the move from **(A)** to **(B)** has a trade-off to pay. Soundness—in the sense of Definition 2.3(1)—is preserved, because the set of computed answers for the implementation **(B)** is a subset of the computed answers for the implementation **(A)**. However, weak completeness—in the sense of Definition 2.3(2)—is not preserved in general, as shown by the following example.

Example 5.4

Let $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$, \mathcal{P} and G be as in Example 2.2. Remember that G'' is as shown in Example 5.1. Then, considering the implementation **(B)** of \sim' for generic proximity relations, **Prolog** only computes the answer $\text{sol}_1 = \langle \sigma_1, \mu_1, \emptyset \rangle$ for G'' . No computed answer subsumes the ground solutions $\text{sol}_2, \text{sol}_3$ of G shown in Example 2.2. **Prolog**'s computation strategy is not responsible for the lack of completeness in this case. \square

Nevertheless, we conjecture that the implementation **(B)** behaves as a flexible restriction of the goal solving system given by the implementation **(A)** in the sense of Definition 2.4. Then, due to Lemma 2.1, we conjecture correctness in the flexible sense for **(B)**. In other words, we claim that our **Prolog**-based system for SQCLP using implementation **(B)** of \sim' is sound and we conjecture that it is also weakly complete in the flexible sense, except for the unavoidable failures caused by **Prolog**'s computation strategy. This conjecture is confirmed as far as the Example 2.2 is concerned, because the computed answer sol_1 subsumes the other ground solutions sol_2 and sol_3 of G in the flexible sense, as shown in the same example.

A further optimization of implementation **(B)** is possible if the given proximity relation \mathcal{S} is transitive — i.e. a similarity. In this case our prototype system implements \sim' by means of the following **Prolog** code:

(C) Practical implementation of \sim' intended for similarity relations.

```

s1  ~'(X,Y,W) :- var(X), var(Y), ~'_v(X,Y,W).
s2  ~'(X,Y,W) :- var(X), nonvar(Y), ~'_v(X,Y,W).
s3  ~'(X,Y,W) :- nonvar(X), var(Y), ~'_v(X,Y,W).
s4  ~'(X,Y,W) :- nonvar(X), nonvar(Y), ~'_c(X,Y,W).

v1  ~'_v(X,Y,W) :- qVal(W), X = Y.

c1  ~'_c(u,u',W) :- qVal(W), qBound(W,t,λ).
c2  ~'_c(c(X1,...,Xn),c'(Y1,...,Yn),W) :- qVal(W), qBound(W,t,λ),
    qVal(W1), qBound(W,t,W1), ~'(X1,Y1,W1),
    ...
    qVal(Wn), qBound(W,t,Wn), ~'(Xn,Yn,Wn).
```

where the only difference w.r.t. implementation **(B)** is that **(C)** uses the predicate call $\sim'_v(X,Y,W)$ instead of $\sim'_c(X,Y,W)$ at the bodies of clauses S_2 and S_3 .

A useful way to understand the difference between **(B)** and **(C)** is to think of both

as different implementations of a unification algorithm modulo a given proximity relation \mathcal{S} . In both cases, a predicate call $\sim'(\mathbf{X}, \mathbf{Y}, \mathbf{W})$ is intended to compute a unifier modulo \mathcal{S} with qualification degree \mathbf{W} for \mathbf{X} and \mathbf{Y} —more precisely, for the terms bound to \mathbf{X} and \mathbf{Y} at run-time—and clauses S_i ($i = 1, 2, 3, 4$) distinguish four possible cases in the same manner. The two implementations differ only in the actions taken in each of these four cases. The actions executed by implementation **(B)** can be intuitively described as follows:

1. *Case 1:* both \mathbf{X} and \mathbf{Y} are variables.
Action: just unify them (achieved by clause V_1).
2. *Case 2:* \mathbf{X} is a variable and \mathbf{Y} is bound to a non-variable term.
Actions: Compute alternative solutions by binding \mathbf{X} to non-variable terms whose root symbol is \mathcal{S} -close to the root symbol of the term bound to \mathbf{Y} (achieved by clauses C_1 and C_2). In particular one of these solutions will correspond to binding \mathbf{X} to the term bound to \mathbf{Y} .
3. *Case 3:* \mathbf{Y} is a variable and \mathbf{X} is bound to a non-variable term.
Actions: Compute alternative solutions by binding \mathbf{Y} to non-variable terms whose root symbol is \mathcal{S} -close to the root symbol of the term bound to \mathbf{X} (achieved by clauses C_1 and C_2). In particular one of these solutions will correspond to binding \mathbf{Y} to the term bound to \mathbf{X} .
4. *Case 4:* both \mathbf{X} and \mathbf{Y} are bound to non-variable terms, both with root and n children terms.
Action: first check that the root symbols of the terms bound to \mathbf{X} and \mathbf{Y} are \mathcal{S} -close; then decompose these two terms and recursively proceed to unify the i -th child of the term bound to \mathbf{X} and the i -th child of the term bound to \mathbf{Y} , for $i = 1 \dots n$ (achieved by clauses C_1 and C_2).

On the other hand, an intuitive description of implementation **(C)** is as follows:

1. *Case 1:* both \mathbf{X} and \mathbf{Y} are variables.
Action: as in case (1) of implementation **(B)**.
2. *Case 2:* \mathbf{X} is a variable and \mathbf{Y} is bound to a non-variable term.
Action: just bind \mathbf{X} to the term bound to \mathbf{Y} (achieved by clause V_1).
3. *Case 3:* \mathbf{Y} is a variable and \mathbf{X} is bound to a non-variable term.
Action: just bind \mathbf{Y} to the term bound to \mathbf{X} (achieved by clause V_1).
4. *Case 4:* both \mathbf{X} and \mathbf{Y} are bound to non-variable terms, both with root and n children terms.
Action: as in case (4) of implementation **(B)**.

Clearly, the difference between these two implementations is limited to cases (2) and (3), where **(B)** enumerates a set of various alternative unifiers while **(C)** behaves in a deterministic way, computing just one of these unifiers. In fact, **(C)** behaves as a **Prolog** implementation of known unification algorithms modulo a given similarity relation \mathcal{S} , as those presented in (Arcelli Fontana and Formato 2002; Sessa 2002) (only for the qualification domain \mathcal{U}) and other related papers, which are complete in the flexible sense for solving unification problems. This is due to the fact that the substitution $\{X \mapsto t\}$ can be taken as the unique unifier

computed for a variable X and a term t , that subsumes in the weak sense other possible unifiers thanks to the transitivity property of \mathcal{S} .

Concerning the behaviour of our Prolog-based SQCLP system when **(C)** is used as the implementation of the \sim' predicate, we claim soundness for any choice of \mathcal{S} (transitive or not), because all the computed answers can be also computed by **(B)**, which is sound. In case that \mathcal{S} is transitive, weak completeness in the flexible sense is the best behaviour that can be expected, but more research is still needed to clarify this issue. The example below shows that weak completeness in the flexible sense generally fails for unification problems (and with more reason for general SQCLP goals), when \mathcal{S} is not transitive. In fact, the same example shows that transitivity of \mathcal{S} is a necessary requirement for the completeness (in the flexible sense) of unification algorithms modulo \mathcal{S} of the kind presented in (Sessa 2002) and related papers.

Example 5.5

Consider for the last time the admissible triple $\langle \mathcal{S}, \mathcal{U}, \mathcal{R} \rangle$ of Example 2.2, the empty program, the goal G shown in Example 5.1, and the CLP goal G'' obtained as translation of G and shown in Example 5.1, which is:

$$\begin{aligned} & qVal(W_1), \quad qBound(0.8, 1, W_1), \quad \sim'(Y, X, W_1), \\ & qVal(W_2), \quad qBound(0.8, 1, W_2), \quad \sim'(X, b, W_2), \\ & qVal(W_3), \quad qBound(0.8, 1, W_3), \quad \sim'(Y, c, W_3) \end{aligned}$$

Note that G is a unification problem modulo \mathcal{S} with the three ground solutions shown in Example 2.2. The proximity relation \mathcal{S} is not transitive, because $\mathcal{S}(b, c) = 0.4 \not\geq 0.9 = \mathcal{S}(b, a) \sqcap \mathcal{S}(a, c)$. The resolution of G'' by using the Prolog code **(C)** for \sim' eventually reduces to solving a new goal of the form

$$qVal(W_3), \quad qBound(0.8, 1, W_3), \quad \sim'(b, c, W_3)$$

which fails, since $\mathcal{S}(b, c) = 0.4 \not\geq 0.8$. In this example, Prolog's computation strategy is not responsible for the lack of completeness. \square

We have just discussed three possible Prolog implementations of the CLP clauses in the set EQ'_S , called **(A)**, **(B)** and **(C)**. The Prolog-based prototype system for SQCLP programming presented in the next subsection only supports implementations **(B)** and **(C)**, using two different predicates *prox/4* and *sim/4*, respectively, to implement the behaviour of \sim' appropriate in each case. By default, the system assumes implementation **(B)**, and a program directive `#optimized_unif` must be used in the case that implementation **(C)** is desired.

5.2 (S)QCLP: A Prototype System for SQCLP Programming

The prototype implementation object of this subsection is publicly available, and can be found at:

<http://gpd.sip.ucm.es/cromdia/qclp>

The system currently requires the user to have installed either *SICStus Prolog* or *SWI-Prolog*, and it has been tested to work under Windows, Linux and MacOSX

platforms. The latest version available at the time of writing this paper is 0.6. If a latter version is available some things might have changed but in any case the main aspects of the system should remain the same. Please consult the *changelog* provided within the system itself for specific changes between versions.

SQCLP is a very general programming scheme and, as such, it supports different proximity relations, different qualification domains and different constraint domains when building specific instances of the scheme for any specific purpose. As it would result impossible to provide an implementation for every admissible triple (or instance of the scheme), it becomes mandatory to decide in advance what specific instances will be available for writing programs in (S)QCLP. In essence:

1. In its current state, the only available constraint domain is \mathcal{R} . Thus, under both *SICStus Prolog* and *SWI-Prolog* the library `clpr` will provide all the available primitives in (S)QCLP programs.
2. The available qualification domains are: ‘b’ for the domain \mathcal{B} ; ‘u’ for the domain \mathcal{U} ; ‘w’ for the domain \mathcal{W} ; and any strict cartesian product of those, as e.g. ‘(u,w)’ for the product domain $\mathcal{U} \otimes \mathcal{W}$.
3. With respect to proximity relations, the user will have to provide, in addition to the two symbols and their proximity value, their *kind* (either predicate or constructor) and their *arity*. Both kind and arity must be the same for each pair of symbols having a proximity value different of **b**.

Note, however, that when no specific proximity relation \mathcal{S} is provided for a given program, \mathcal{S}_{id} is then assumed. Under this circumstances, an obvious technical optimization consists in transforming the original program only with $\text{elim}_{\mathcal{D}}$, thus reducing the overload introduced in this case by $\text{elim}_{\mathcal{S}}$. The reason behind this optimization is that for any given SQCLP($\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , it is also true that \mathcal{P} is a QCLP(\mathcal{D}, \mathcal{C})-program, therefore $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ must semantically be equivalent to $\text{elim}_{\mathcal{D}}(\mathcal{P})$. Nevertheless, $\text{elim}_{\mathcal{D}}(\mathcal{P})$ behaves more efficiently than $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ due to the reduced number of resulting clauses. Thus, in order to improve the efficiency, the system will avoid the use of $\text{elim}_{\mathcal{S}}$ when no proximity relation is provided by the user.

The final available instances in the (S)QCLP system are: SQCLP($\mathcal{S}, \mathbf{b}, \text{clpr}$), SQCLP($\mathcal{S}, \mathbf{u}, \text{clpr}$), SQCLP($\mathcal{S}, \mathbf{w}, \text{clpr}$), SQCLP($\mathcal{S}, (\mathbf{u}, \mathbf{w}), \text{clpr}$), ... and their counterparts in the QCLP scheme when $\mathcal{S} = \mathcal{S}_{\text{id}}$.

5.2.1 Programming in (S)QCLP

Programming in (S)QCLP is straightforward if the user is accustomed to the Prolog programming style. However, there are three syntactic differences with pure Prolog:

1. Clauses implications are replaced by “<-d-” where $d \in D \setminus \{\mathbf{b}\}$. If $d = \mathbf{t}$, then the implication can become just “<-”. E.g. “<-0.9-” is a valid implication in the domains \mathcal{U} and \mathcal{W} ; and “<-(0.9,2)-” is a valid implication in the domain $\mathcal{U} \otimes \mathcal{W}$.
2. Clauses in (S)QCLP are not finished with a dot (.). They are separated by

layout, therefore all clauses in a (S)QCLP program must start in the same column. Otherwise, the user will have to explicitly separate them by means of semicolons (;).

3. After every body atom (even constraints) the user can provide a threshold condition using '#'. The notation '?' can also be used instead of some particular qualification value, but in this case the threshold condition '#?' can be omitted.

Comments are as in Prolog:

```
% This is a line comment.
/* This is a multi-line comment, /* and they nest! */. */
```

and the basic structure of a (S)QCLP program is the following (line numbers are for reference):

File: *Peano.qclp*

```
1 % Directives...
2 # qdom w

3 % Program clauses...
4 % num( ?Num )
5 num(z) <--
6 num(s(X)) <-1- num(X)
```

In the previous small program, lines 1, 3 and 4 are line comments, line 2 is a program directive telling the compiler the specific qualification domain the program is written for, and lines 5 and 6 are program clauses defining the well-known Peano numbers. As usual, comments can be written anywhere in the program as they will be completely ignored (remember that a line comment must necessarily end in a new line character, therefore the very last line of a file cannot contain a line comment), and directives must be declared before any program clause. There are three program directives in (S)QCLP:

1. The first one is “#qdom *qdom*” where *qdom* is any system available qualification domain, i.e. **b**, **u**, **w**, (**u,w**)... See line 2 in the previous program sample as an example. This directive is mandatory because the user must tell the compiler for which particular qualification domain the program is written.
2. The second one is “#prox *file*” where *file* is the name of a file (with extension **.prox**) containing a proximity relation. If the name of the file starts with a capital letter, or it contains spaces or any special character, *file* will have to be quoted with single quotes. For example, assume that with our program file we have another file called *Proximity.prox*. Then, we would have to write “#prox 'Proximity'” to link the program with such proximity relation. This directive is optional, and if omitted, the system assumes that the program is of an instance of the QCLP scheme.
3. The third one is “#optimized_unif”. This directive tells the compiler that the program is intended to be used with the implementation **(C)** for the predicate \sim' , as explained in Subsection 5.1.3.

Proximity relations are defined in files of extension `.prox` with the following form:

File: *Work.prox*

```

1  % Predicates: pprox( S1, S2, Arity, Value ).
2  pprox(wrote, authored, 2, (0.9,0)).

3  % Constructors: cprox( S1, S2, Arity, Value ).
4  cprox(king_lear, king_liar, 0, (0.8,2)).
```

where the file can contain `pprox/4` Prolog facts, for defining proximity between predicate symbols of any arity; or `cprox/4` Prolog facts, for defining proximity between constructor symbols of any arity. The arguments of both `pprox/4` and `cprox/4` are: the two symbols, their arity and its proximity value. Note that, although it is not made explicit the qualification domain this proximity relation is written for, all values in it must be of the same specific qualification domain, and this qualification domain must be the same declared in every program using the proximity relation. Otherwise, the solving of equations may produce unexpected results or even fail.

Reflexive and symmetric closure is inferred by the system, therefore, there is no need for writing reflexive proximity facts, nor the symmetric variants of proximity facts already provided. You can notice this in the previous sample file in which neither reflexive proximity facts, nor the symmetric proximity facts to those at lines 2 and 4 are provided. In the case of being explicitly provided, additional (repeated) solutions might be computed for the same given goal, although soundness and weak completeness of the system should still be preserved. Transitivity is neither checked nor inferred so the user will be responsible for ensuring it if desired.

As the reader would have already guessed, the file `Work.prox` implements the proximity relation \mathcal{S}_r of Example 4.1 in (S)QCLP. Finally, the program \mathcal{P}_r of Example 4.1 can be represented in (S)QCLP as follows:

File: *Work.qclp*

```

1  # qdom (u,w)
2  # prox 'Work'

3  % famous( ?Author )
4  famous(shakespeare) <-(0.9,1)-

5  % wrote( ?Author, ?Book )
6  wrote(shakespeare, king_lear) <-(1,1)-
7  wrote(shakespeare, hamlet) <-(1,1)-

8  % good_work( ?Work )
9  good_work(X) <-(0.75,3)- famous(Y)#(0.5,100), authored(Y,X)
```

Note that, at line 1 the qualification domain $\mathcal{U} \otimes \mathcal{W}$ is declared, and at line 2 the proximity relation at `Work.prox` is linked to the program. In addition, observe that one threshold constraint is imposed for a body atom in the program clause at line 9, effectively requiring to prove `famous(Y)` for a qualification value of *at least* $(0.5, 100)$ to be able to use this program clause.

Finally, we explain how constraints are written in (S)QCLP. As it has already been said, only \mathcal{R} is available, thus both in *SICStus Prolog* and *SWI-Prolog* the

library `clpr` is the responsible for providing the available primitive predicates. Given that constraints are primitive atoms of the form $\mathbf{r}(\bar{\mathbf{t}}_n)$ where $\mathbf{r} \in PP^n$ and \mathbf{t}_i are terms; primitive atoms share syntax with usual Prolog atoms. At this point, and having that many of the primitive predicates are syntactically operators (hence not valid identifiers), the syntax for predicate symbols has been extended to include operators, therefore predicate symbols like $op_+ \in PP^3$, which codifies the operation $+$ in a 3-ary predicate, will let us to build constraints of the form $+(A,B,C)$, that must be understood as in $A + B = C$ or $C = A + B$. Similarly, predicate symbols like $cp_> \in PP^2$, which codifies the comparison operator $>$ in a binary predicate, will let us to build constraints of the form $>(A,B)$, that must be understood as in $A > B$. Any other primitive predicate such as $maximize \in PP^1$, will let us to build constraints like $maximize(X)$. Valid primitive predicate symbols include $+$, $-$, $*$, $/$, $>$, $>=$, $=<$, $<$, `maximize`, `minimize`, etc.

Threshold constraints can also be provided for primitive atoms in the body of clauses with the usual notation. Note, however, that due the semantics of `SQCLP`, all primitive atoms can be trivially proved with `t` if they ever succeeds—so threshold constraints become, in this case, of no use.

The syntax for constraints explained above follows the standard syntax for atoms. Nonetheless, the system also allows to write these constraints in a more natural infix notation. More precisely, $+(A,B,C)$ can be also written in the infix form $A+B=C$ or $C=A+B$, and $>(X,Y)$ in the infix form $X>Y$; and similarly for other *op* and *cp* constraints. When using infix notation, threshold conditions can be set by (optionally) enclosing the primitive atom between parentheses, therefore becoming $(A+B=C)\#t$, $(C=A+B)\#t$ or $(X>Y)\#t$ (or any other valid qualification value or ‘?’). Using parentheses is recommended to avoid understanding that the threshold condition is set only for the last term in the constraint, which would make no sense. Note that even in infix notation, operators cannot be nested, that is, terms A , B , C , X and Y cannot have operators as main symbols (neither in prefix nor in infix notation), so the infix notation is just a syntactic sugar of its corresponding prefix notation.

As a final example for constraints, one could write the predicate `double/2` in `(S)QCLP`, for computing the double of any given number, with just the clause `double(N,D) <-- *(N,2,D)`, or `double(N,D) <-- N*2=D` for a clause with a more natural syntax.

5.2.2 The interpreter for `(S)QCLP`

The interpreter for `(S)QCLP` has been implemented on top of both *SICStus Prolog* and *SWI-Prolog*. To load it, one must first load her desired (and supported) Prolog system and then load the main file of the interpreter—i.e. `qclp.pl`—, that will be located in the main `(S)QCLP` folder among other folders. Once loaded, one will see the welcome message and will be ready to compile and load programs, and to execute goals.

WELCOME TO `(S)QCLP 0.6`

`(S)QCLP` is free software and comes with absolutely no warranty.

Support & Updates: <http://gpd.sip.ucm.es/cromdia/qclp>.

Type `:help.` for help.

```
yes
| ?-
```

From the interpreter for (S)QCLP one can, in addition to making use of any standard Prolog goals, use the specific (S)QCLP commands required for both interacting with the (S)QCLP system, and for compiling/loading SQCLP programs. All these commands take the form:

`:command.`

if they do not require arguments, or:

`:command(Arg1, ..., Argn).`

if they do; where each argument *Arg*_{*i*} must be a Prolog atom unless stated otherwise. The most useful commands are:

- `:cd(Folder).`
Changes the working directory to *Folder*. *Folder* can be an absolute or relative path.
- `:compile(Program).`
Compiles the (S)QCLP program '*Program.qclp*' producing the equivalent Prolog program in the file '*Program.pl*'.
- `:load(Program).`
Loads the already compiled (S)QCLP program '*Program.qclp*' (note that the file '*Program.pl*' must exist for the program to correctly load).
- `:run(Program).`
Compiles the (S)QCLP program '*Program.qclp*' and loads it afterwards. This command is equivalent to executing: `:compile(Program), :load(Program).`

For illustration purposes, we will assume that you have the files `Work.prox` and `Work.qclp` (both as seen before) in the folder `~/examples`. Under these circumstances, after loading your preferred Prolog system and the interpreter for (S)QCLP, one would only have to change the working directory to that where the files are located:

```
| ?- :cd('~/examples').
```

and run the program:

```
| ?- :run('Work').
```

If no errors are encountered, one should see the output:

```
| ?- :run('Work').
<Work> Compiling...
<Work> QDom: 'u,w'.
<Work> Prox: 'Work'.
```

```

<Work> Translating to QCLP...
<Work> Translating to CLP...
<Work> Generating code...
<Work> Done.
<Work> Loaded.
yes

```

and now everything is ready to execute goals for the program loaded.

5.2.3 Executing SQCLP-Goals

Recall that goals have the form $A_1\#W_1, \dots, A_m\#W_m \parallel W_1 \triangleright^? \beta_1, \dots, W_m \triangleright^? \beta_m$ which in actual (S)QCLP syntax becomes:

```
| ?- A1#W1, ..., Am#Wm :: W1 >= B1, ..., Wm >= Bm.
```

Note the following:

1. Goals must end in a dot (.).
2. The symbol ‘ \parallel ’ is replaced by ‘ $::$ ’.
3. The symbol ‘ $\triangleright^?$ ’ is replaced by ‘ $>=$ ’ (and this is independent of the qualification domain in use, so that it may mean \leq in \mathcal{W}).
4. Conditions of the form $W \triangleright^? ?$ *must be omitted*, therefore $A_1\#W_1, A_2\#W_2 \parallel W_1 \triangleright^? ?, W_2 \triangleright^? \beta_2$ becomes “ $A1\#W1, A2\#W2 :: W2 >= B2.$ ”, and $A\#W \parallel W \triangleright^? ?$ becomes just “ $A\#W.$ ”.

Assuming now that we have loaded the program `Work.qclp` as explained before, we can execute the goal $good_work(king_liar)\#W \parallel W \triangleright^? (0.5, 100)$:

```

| ?- good_work(king_liar)#W::W>=(0.5,10).
W = (0.6,5.0) ? ;
W = (0.675,4.0) ? ;
no

```

Note that the system computes two answers, with different qualification values. In this simple example, the second computed answer provides a better qualification value. In general, different computed answers for the same goal come with different qualification values and it is not always the case that one of the answers provides the optimal qualification value.

5.2.4 Examples

To finish this subsection, we are now showing some additional goal executions using the interpreter for (S)QCLP and the programs displayed along the paper.

Peano. Consider the program `Peano.qclp` as displayed at the beginning of Subsection 5.2.1. Qualifications in this program are intended as a cost measure for obtaining a given number in the Peano representation, assuming that each use of the clause at line 6 requires to pay *at least* 1. In essence, threshold conditions will

impose an upper bound over the maximum number obtainable in goals containing the atom `num(X)`. Therefore if we ask for numbers *up to* a cost of 3 we get the following answers:

```
Goal    ?- num(X)#W::W>=3.
Sol1    W = 0.0, X = z ? ;
Sol2    W = 1.0, X = s(z) ? ;
Sol3    W = 2.0, X = s(s(z)) ? ;
Sol4    W = 3.0, X = s(s(s(z))) ? ;
no
```

Work. Consider now the program `Work.qclp` and the proximity relation `Work.prox`, both as displayed in Subsection 5.2.1 above. In this program, qualifications behave as the conjunction of the certainty degree of the user confidence about some particular atom, and a measure of the minimum cost to pay for proving such atom. In these circumstances, we could ask—just for illustration purposes—for famous authors with a minimum certainty degree—for them being actually famous—of 0.5, and with a proof cost of no more than 30 (think of an upper bound for possible searches in different databases). Such a goal would have, in this very limited example, only the following solution:

```
Goal    ?- famous(X)#W::W>=(0.5,30).
Sol1    W = (0.9,1.0), X = shakespeare ? ;
no
```

meaning that we can have a confidence of `shakespeare` being famous of 0.9, and that we can prove it with a cost of 1.

Now, in a similar fashion we could try to obtain different works that can be considered as good works by using the last clause in the example. Limiting the search to those works that can be considered good with a qualification value better or equal to (0.5,100) produce the following result:

```
Goal    ?- good_work(X)#W::W>=(0.5,100).
Sol1    W = (0.675,4.0), X = king_lear ? ;
Sol2    W = (0.6,5.0), X = king_liar ? ;
no
```

A valid ground answer for this goal is $gsol = \langle \eta, \rho, \emptyset \rangle$ where $\eta = \{X \mapsto king_liar\}$ and $\rho = \{W \mapsto (0.675, 4)\}$ (which corresponds to the second computed answer for the ground goal displayed in Subsection 5.2.3). Note that the first computed answer shown above is $ans = \langle \sigma, \mu, \emptyset \rangle$ where $\sigma = \{X \mapsto king_lear\}$ and $\mu = \{W \mapsto (0.675, 4)\}$ which subsumes $gsol$ in the flexible sense via $\nu = \varepsilon \in \text{Sol}_{\mathcal{R}}(\emptyset)$.

Library. Finally, consider the program \mathcal{P}_s and the proximity relation \mathcal{S}_s , both as displayed in Figure 1 of Section 2. As it has been said when this example was introduced, the predicate `guessRdrLvl` takes advantage of attenuation factors to encode heuristic rules to compute reader levels on the basis of vocabulary level and other book features. As an illustration of use, consider the following goal:

```

Goal    ?- guessRdrLvl(book(2, 'Dune', 'F. P. Herbert', english, sciFi,
                        medium, 345), Level)#W.

Sol1   W = 0.8, Level = intermediate ? ;
        ...
Sol6   W = 0.7, Level = upper ?
        yes

```

Here we ask for possible ways of classifying the second book in the library according to reader levels. We obtain as valid solutions, among others, **intermediate** with a certainty factor of 0.8; and **upper** with a certainty factor of 0.7. These valid solutions show that the predicate *guessRdrLvl* tries with different levels for any certain book based on the heuristic implemented by the qualified clauses.

To conclude, consider now the goal proposed in Section 2 for this program. For such goal we obtain:

```

Goal    ?- search(german, essay, intermediate, ID)#W::W>=0.65.

Sol1   W = 0.8, ID = 4 ?
        yes

```

What tells us that the forth book in the library is written in German, it can be considered to be an essay, and it is targeted for an intermediate reader level. All this with a certainty degree of *at least* 0.8.

5.3 Efficiency

The minimum—and unavoidable—overload introduced by qualifications and proximity relations in the transformed programs manifests itself in the case of (S)QCLP programs which use the identity proximity relation and have **t** as the attenuation factor of all their clauses. In order to measure this overload we have made some experiments using some program samples, taken from the *SICStus Prolog Benchmark* that can be found in:

<http://www.sics.se/isl/sicstuswww/site/performance.html>

and we have compared the time it took to repeatedly execute a significant number of times each program in both (S)QCLP and *SICStus Prolog* making use of a *slightly* modified (to ensure a correct behaviour in both systems) version of the harness also provided in the same site.

From all the programs available in the aforementioned site, we selected the following four:

- *naivrev*: naïve implementation of the predicate that reverses the contents of a list.
- *deriv*: program for symbolic derivation.
- *qsort*: implementation of the well-known sorting algorithm *Quicksort*.
- *query*: obtaining the population density of different countries.

No other program could be used because they included impure features such as cuts which are not currently supported by our system. In order to adapt these Prolog programs to our setting the following modifications were required:

1. All the program clause are assumed to have **t** as attenuation factor. After including these attenuation factors, we obtain as results QCLP programs. More specifically we obtain two QCLP programs for each initial Prolog program, one using the qualification domain \mathcal{B} (because this domain uses trivial constraints), and another using the qualification domain \mathcal{U} (which uses \mathcal{R} -constraints).
2. We define an empty proximity relation, allowing us to obtain two additional SQCLP-programs.
3. By means of the program directive “**#optimized_unif**” defined in Subsection 5.2.1, each SQCLP program can be also executed in this optimized mode. Therefore each original Prolog Program produces six (S)QCLP programs, denoted as Q(b), Q(u), PQ(b), PQ(u), SQ(b) and SQ(u) in Table 1.

Additionally some minor modifications to the program samples have been introduced for compatibility reasons, i.e. additions using the predicate **is/2** were replaced, both in the Prolog version of the benchmark and in the multiple (S)QCLP versions, by **clpr** constraints. In any case, all the program samples used for this benchmarks in this subsection can be found in the folder **benchmarks/** of the (S)QCLP distribution.

Finally, we proceeded to solve the same goals for every version of the benchmark programs, both in *SICStus Prolog* and in (S)QCLP. The benchmark results can be found in Table 1. All the experiments were performed in a computer with a Intel(R) Core(TM)2 Duo CPU at 2.19GHz and with 3.5 GB RAM.

Table 1. Time overload factor with respect to Prolog

Program	Q(b) ^a	Q(u) ^b	PQ(b) ^c	PQ(u) ^d	SQ(b) ^e	SQ(u) ^f
naivrev	1.80	10.71	4289.79	4415.11	56.22	65.75
deriv	1.94	10.60	331.45	469.67	29.63	39.32
qsort	1.05	1.11	135.59	136.98	2.51	2.83
query	1.02	1.12	7.17	7.13	3.80	3.88

^a QCLP(\mathcal{B}, \mathcal{R}) version (i.e. the program does not have the **#prox** directive).

^b QCLP(\mathcal{U}, \mathcal{R}) version (i.e. the program does not have the **#prox** directive).

^c SQCLP($\mathcal{S}_{id}, \mathcal{B}, \mathcal{R}$) version.

^d SQCLP($\mathcal{S}_{id}, \mathcal{U}, \mathcal{R}$) version.

^e SQCLP($\mathcal{S}_{id}, \mathcal{B}, \mathcal{R}$) version with directive **#optimized_unif**.

^f SQCLP($\mathcal{S}_{id}, \mathcal{U}, \mathcal{R}$) version with directive **#optimized_unif**.

The results in the table indicate the slowdown factor obtained for each version of each program. For instance, the first column indicates that the time required for evaluating the goal corresponding to the sample program *naivrev* in QCLP(\mathcal{B}, \mathcal{R}) is about 1.80 times the required time for the evaluation of the same goal in Prolog. Next we discuss the results:

- *Influence of the qualification domain.* In general the difference between the slowdown factors obtained for the two considered qualification domains is not

large. However, in the case of QCLP-programs *naivrev* and *deriv* the difference increases notably. This is due to the different ratios of the \mathcal{B} -constraints w.r.t. the program and \mathcal{U} -constraints w.r.t. the program. It must be noticed that the transformed programs are the same in both cases, but for the implementation of `qval` and `qbound` constraints, which is more complex for \mathcal{U} as one can see in Subsection 5.1. In the case of *naivrev* and *deriv* this makes a big difference because the number of computation steps directly required by the programs is much smaller than in the other cases. Thus the slowdown factor becomes noticeable for the qualification domain \mathcal{U} in computations which require a large number of steps.

- *Influence of the proximity relation.* The introduction of a proximity relation—even the identity—is very significative, since unification in the original **Prolog** program is handled by calls to the predicate \sim' in the SQCLP program. This is particularly relevant when the computation introduces large constructor terms, as in the case of *naivrev* which deals with **Prolog** lists. The efficient Prolog unification is replaced by an explicit term decomposition.
- *Influence of the optimized unification.* As seen in the table, the use of the program directive `#optimized_unif` causes a clear increase in the efficiency of goal solving for these examples. This is due to the use of the implementation **(C)** for the predicate \sim' instead of the implementation **(B)** (see Subsection 5.1). The speed-up is especially noticeable when large data structures are involved in the unification as can be seen for the sample programs *naivev* and *deriv*. The reason is that the implementation **(C)** avoids costly term decompositions required by the other implementation.

6 Conclusions

In our recent work (Rodríguez-Artalejo and Romero-Díaz 2010a) we extended the classical CLP scheme to a new programming scheme SQCLP whose instances $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ were parameterized by a proximity relation \mathcal{S} , a qualification domain \mathcal{D} and a constraint domain \mathcal{C} . This new scheme offered extra facilities for dealing with expert knowledge representation and flexible query answering. In this paper we have set the basis for a practical use of SQCLP by providing a prototype implementation on top CLP(\mathcal{R}) systems like *SICStus Prolog* and *SWI-Prolog*, based on semantically correct program transformation techniques and supporting several interesting instances of the scheme.

The transformation techniques presented in Section 4 work over programs and goals in two steps, formalized as the composition of two transformations: $\text{elim}_{\mathcal{S}}$ and $\text{elim}_{\mathcal{D}}$. Our mathematical results show that $\text{elim}_{\mathcal{S}}$ replaces the explicit use of a proximity relation by using just qualification values and clause annotations, which are in turn replaced by purely CLP computations thanks to $\text{elim}_{\mathcal{D}}$. The composed effect of the two transformations ultimately enables to solve goals for SQCLP programs by applying any capable CLP goal solving system to their CLP translations.

The prototype implementation presented in Section 5 relies on the transformation

techniques, improved with some optimizations. It has finally allowed us to execute all the examples shown in this paper—and in previous ones—, and a series of benchmarks for measuring the overload actually introduced by proximity relations—or by similarity relations—and by clause annotations and qualifications. While we are aware that the prototype implementation presented in this paper has to be considered a research tool (and as such, we admit that it cannot be used for industrial applications), we think that it can contribute to the field as a quite solid implementation of an extension of $\text{CLP}(\mathcal{R})$ with proximity relations and qualifications.

Some related implementation techniques and systems have been presented in the Introduction. However, as far as we know, no other implementation in this field has ever provided simultaneous support for proximity (and similarity) relations, qualifications via clause annotations and $\text{CLP}(\mathcal{R})$ style programming. Moreover, the development of our prototype has used both semantically correct methods and careful optimizations, aiming at a balance between theoretical foundations and a sound but practical system.

In the future, and taking advantage of the prototype system we have already developed, we plan to investigate possible applications which can profit from proximity relations and qualifications, such as in the area of flexible query answering. In particular, we plan to investigate application related to flexible answering of queries to XML documents, in the line of (Campi et al. 2009) and other related papers.

As support for practical applications, we also plan to increase the repertoire of constraint and qualification domains which can be used in the (S)QCLP prototype, adding the constraint domain \mathcal{FD} and the qualification domain \mathcal{W}_d defined in Section 2.2.3 of (Rodríguez-Artalejo and Romero-Díaz 2010b). On a more theoretical line, other possible lines of future work include: a) investigation of unification modulo a given proximity relation \mathcal{S} , not assuming transitivity for \mathcal{S} and proving soundness and completeness properties for the resulting unification algorithm; b) building upon (a), extension of the $\text{SLD}(\mathcal{D})$ resolution procedure presented in (Rodríguez-Artalejo and Romero-Díaz 2008) to a SQCLP goal solving procedure able to work with constraints and a proximity relation, including also soundness and completeness proofs; and c) extension of the QCFLP (*qualified constraint functional logic programming*) scheme in (Caballero et al. 2009) to work with a proximity relation and higher-order functions, as well as the implementation of the resulting scheme in the CFLP(\mathcal{C})-system Toy (Arenas et al. 2007).

Acknowledgements

The authors would like to thank to the anonymous reviewers, whose detailed and constructive comments and suggestions helped to revise and improve the paper; and to Jesús Almendros for pointing to bibliographic references in the area of flexible query answering.

References

- APT, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal Models and Semantics. Elsevier and The MIT Press, 493–574.
- ARCELLI FONTANA, F. 2002. Likelog for flexible query answering. *Soft Computing* 7, 107–114.
- ARCELLI FONTANA, F. AND FORMATO, F. 1999. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied computing (SAC'99)*. ACM Press, New York, NY, USA, 260–267.
- ARCELLI FONTANA, F. AND FORMATO, F. 2002. A similarity-based resolution rule. *International Journal of Intelligent Systems* 17, 9, 853–872.
- ARENAS, P., FERNÁNDEZ, A. J., GIL, A., LÓPEZ-FRAGUAS, F. J., RODRÍGUEZ-ARCALEJO, M., AND SÁENZ-PÉREZ, F. 2007. *TOY*, a multiparadigm declarative language (version 2.3.1). In R. Caballero and J. Sánchez, editors, *User Manual*, available at <http://toy.sourceforge.net>.
- BALDWIN, J. F., MARTIN, T., AND PILSWORTH, B. 1995. *Fril-Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons.
- BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 2001. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems* 3, 1 (January), 1–29.
- CABALLERO, R., RODRÍGUEZ-ARCALEJO, M., AND ROMERO-DÍAZ, C. A. 2008. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming*. ACM, Valencia, Spain, 185–194.
- CABALLERO, R., RODRÍGUEZ-ARCALEJO, M., AND ROMERO-DÍAZ, C. A. 2009. Qualified computations in functional logic programming. In *Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. LNCS, vol. 5649. Springer-Verlag Berlin Heidelberg, Pasadena, CA, USA, 449–463.
- CAMPI, A., DAMIANI, E., GUINEA, S., MARRARA, S., PASI, G., AND SPOLETINI, P. 2009. A fuzzy extension of the XPath query language. *Journal of Intelligent Information Systems* 33, 3 (December), 285–305.
- DUBOIS, D. AND PRADE, H. 1980. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, NY, USA.
- FREUDER, E. C. AND WALLACE, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58, 1–3, 21–70.
- GEORGET, Y. AND CODOGNET, P. 1998. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*. LNCS, vol. 1520. Springer-Verlag, 205–219.
- GERLA, G. 2001. *Fuzzy logic: mathematical tools for approximate reasoning*. Kluwer Academic Publishers, Norwell, MA, USA.
- GUADARRAMA, S., MUÑOZ, S., AND VAUCHERET, C. 2004. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems* 144, 1, 127–150.
- HÁJEK, P. 1998. *Metamathematics of Fuzzy Logic*. Dordrecht: Kluwer.
- HÖHFELD, M. AND SMOLKA, G. 1988. Definite relations over constraint languages. Tech. Rep. LILOG Report 53, IBM Deutschland.
- ISHIZUKA, M. AND KANAI, N. 1985. Prolog-ELF incorporating fuzzy logic. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, A. K. Joshi, Ed. Morgan Kaufmann, Los Angeles, CA, USA, 701–703.

- JAFFAR, J. AND LASSEZ, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*. ACM New York, NY, USA, Munich, West Germany, 111–119.
- JAFFAR, J., MAHER, M., MARRIOTT, K., AND STUCKEY, P. J. 1998. Semantics of constraints logic programs. *Journal of Logic Programming* 37, 1-3, 1–46.
- JULIÁN, R., MORENO, G., AND PENABAD, J. 2009. An improved reductant calculus using fuzzy partial evaluation techniques. *Fuzzy Sets and Systems* 160, 2, 162–181.
- JULIÁN-IRANZO, P., RUBIO, C., AND GALLARDO, J. 2009. Bousi~Prolog: a prolog extension language for flexible query answering. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 131–147.
- JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2009a. A declarative semantics for Bousi~Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*. ACM, Coimbra, Portugal, 149–160.
- JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2009b. A similarity-based WAM for Bousi~Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009)*. LNCS, vol. 5517. Springer Berlin / Heidelberg, Salamanca, Spain, 245–252.
- KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming* 12, 3&4, 335–367.
- LEE, R. C. T. 1972. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery (ACM)* 19, 1 (January), 109–119.
- LI, D. AND LIU, D. 1990. *A Fuzzy Prolog Database System*. John Wiley & Sons.
- LLOYD, J. W. 1987. *Foundations of Logic Programming, Second Edition*. Springer.
- LOIA, V., SENATORE, S., AND SESSA, M. I. 2004. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems* 144, 1, 151–171.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2001a. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, T. Eiter, W. Faber, and M. Truszczyński, Eds. LNAI, vol. 2173. Springer-Verlag, 351–364.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2001b. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence (EPIA'01)*, P. Brazdil and A. Jorge, Eds. LNAI, vol. 2258. Springer-Verlag, 290–297.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2004. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems* 146, 43–62.
- RIEZLER, S. 1998. Probabilistic constraint logic programming. Ph.D. thesis, Neuphilologischen Fakultät der Universität Tübingen.
- RODRÍGUEZ-ARCALEJO, M. AND ROMERO-DÍAZ, C. A. 2008. Quantitative logic programming revisited. In *Functional and Logic Programming (FLOPS'08)*, J. Garrigue and M. Hermenegildo, Eds. LNCS, vol. 4989. Springer-Verlag, Ise, Japan, 272–288.
- RODRÍGUEZ-ARCALEJO, M. AND ROMERO-DÍAZ, C. A. 2010a. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue* 10, 4–6, 627–642.
- RODRÍGUEZ-ARCALEJO, M. AND ROMERO-DÍAZ, C. A. 2010b. Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity. Tech. Rep. SIC-1-10 (CoRR abs/1009.1977), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain.
- SESSA, M. I. 2001. Translations and similarity-based logic programming. *Soft Computing* 5, 2.

- SESSA, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science* 275, 1-2, 389–426.
- SICS AB. 2010. SICStus Prolog. Available at <http://www.sics.se/sicstus>.
- SWI-PROLOG. 2010. Available at <http://www.swi-prolog.org>.
- VAN EMDEN, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming* 3, 1, 37–53.
- VOJTÁŠ, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems* 124, 361–370.
- ZADEH, L. A. 1965. Fuzzy sets. *Information and Control* 8, 3, 338–353.
- ZADEH, L. A. 1971. Similarity relations and fuzzy orderings. *Information Sciences* 3, 2, 177–200.